МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА «КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ»

МЕТОДИЧНІ ВКАЗІВКИ

по дисциплини "Захист інформації в КС".
Конспект лекцій
(опорний).

для студентів спеціальностей
7.091501: "Комп'ютерні системи та мережі"
7.091502: "Системне програмування"

Затверджено
на засіданні кафедри КІ
Протокол № 1 від 30.08.2010

Рекомендовано до видання
методичною комісією
спеціальностей 7.091501 і 7.091502
Протокол № від

Донецьк — 200_ р.

Навчальне видання

Методичні вказівки
з дисципліни “Захист інформації в КС”

Конспект лекцій.

УДК 681.073

МЕТОДИЧНІ ВКАЗІВКИ ПО ДИСЦИПЛИНИ “ЗАХИСТ ІНФОРМАЦІІ В КС” (конспект лекцій) для студентів спеціальностей "Комп'ютерні системи та мережі" ”Системне програмування” (для студентів очної, заочної та очно – заочної форми навчання).

В курсі **“ЗАХИСТ ІНФОРМАЦІІ В КС”** студенти спрямовані на опанування основних принципів, методів, систем і засобів, що складають захист інформації: теоретичні основи і формальні моделі захисту інформації; способи оцінки захищених систем; правове регулювання в області захисту інформації; засоби проектування і реалізації програмно-апаратних засобів захисту програм і даних; методи застосування програмно-апаратних засобів захисту інформації в операційних системах і обчислювальних мережах; криптографічні методи захисту інформації.

В кожному розділі розглядаються можливі практичного застосування, в основному до проблем інформатики. В усіх розділах приділяється значна увага побудові алгоритмів для розв'язування задач. Поняття, факти, алгоритми, що вивчаються у курсі "Дискретна математика" використовують знання набуті у курсах " Програмування", "Дискретна математика", "Системне програмування", "Мікропроцесорні системи", а також багатьох спеціальних курсах.

Укладачі: А.Ю. Іванов, ст. викл., Ю.А. Іванов, асс.

Рецензент: Мальчева Р.В., к.т.н., доц.
Зорі С. А., к.т.н., доц.

ЛІТЕРАТУРА.

1. Барсуков В.С., Дворянкин С.В., Шеремет И.А. Безопасность связи в каналах телекоммуникаций. // ТЭК. - 1992. - Т. 20. - 122 с.
2. Атака через Интернет /И.Д.Медведовский, П.В.Семьянов, В.В.Платонов; Под ред. проф. П.Д.Зегжды. - СПб.: Мир и семья-95, 1997. - 296 с.
3. Безруков Н.Н. Классификация компьютерных вирусов MS-DOS и методы защиты от них.- М.: Изд. СП “ІСЕ”, 1990.- 48 с.
4. Безруков Н.Н. Компьютерная вирусология. - К.: Укр. советская энцикл. - 1991. - 416 с.
5. Грушо А.А., Тимонина Е.Е. Теоретические основы защиты информации. - М.: Яхтсмен, 1996. - 190 с.
6. Герасименко В.А. Защита информации в автоматизированных системах обработки данных. В 2-х кн. - М.: Энергоатомиздат, 1994. - Кн. 1. - 400 с., кн.2. - 175 с.
7. Жельников В. Криптография от папируса до компьютера. - М.: ABF, 1996. - 336 с.
8. Касперский Е. Компьютерные вирусы в MS-DOS. - М.: Эдель, 1992. - 175 с.
9. Мафтик. С. Механизмы защиты в сетях ЭВМ: Пер. с англ. - М.: Мир, 1993. - 216 с.
10. Моисеенков И. Американская классификация и принципы оценивания безопасности компьютерных систем. // Компьютер Пресс. - 1992. - № 2. - С. 61-68; № 3. - С. 47-54.
11. Расторгуев С. Программные методы защиты информации в компьютерных сетях. - М.: Яхтсмен, 1993. - 187 с.
12. Теория и практика обеспечения информационной безопасности. / Под ред. П.Д.Зегжды. - М.: Яхтсмен, 1996. - 192 с.

13. Фролов А.В., Фролов Г.В. Осторожно: компьютерные вирусы. - М.: ДИАЛОГ-МИФИ, 1996. - 256 с.

Лекція 1. Введення в інформаційну безпеку

1. Вступ.

Використання комп'ютерів і автоматизованих технологій привело до появи нових проблем. Комп'ютери, об'єднані в мережі, можуть надавати доступ до великої кількості різних даних. Це збільшує число комп'ютерних злочинів, що може привести в кінцевому рахунку до підриву економіки. І тому ясно, що інформація - це ресурс, якому треба захищати.

Число комп'ютерних злочинів росте - також збільшуються масштаби комп'ютерних зловживань. По оцінці фахівців США, збиток від комп'ютерних злочинів збільшується на 35 відсотків у рік і складає близько 3.5 мільярдів доларів. Однієї з причин є сума грошей, одержувана в результаті злочину: у той час як збиток від середнього комп'ютерного злочину складає 560 тисяч доларів, при пограбуванні банку - усього лише 19 тисяч доларів.

З іншої сторони шансів бути пійманим у комп'ютерного злочинця набагато менше, ніж у грабіжника банку - і навіть при пійманні в нього менше шансів потрапити у в'язницю. Виявляється в середньому 1 відсоток комп'ютерних злочинів. І імовірність того, що за комп'ютерне шахрайство злочинець потрапить у в'язницю, менше 10 відсотків.

Головною причиною наявності втрат, зв'язаних з комп'ютерами, є недостатня освіченість в області безпеки. Тільки наявність знань в області безпеки може припинити інциденти і помилки, забезпечити ефективне застосування мір захисту, запобігти чи злочин своєчасний знайти порушення.

Поінформованість кінцевого користувача про міри безпеки забезпечує чотири рівні захисту комп'ютерних і інформаційних ресурсів:

1. Запобігання - тільки авторизований персонал має доступ до інформації і технології

2. Виявлення - забезпечується раннє виявлення злочинів і зловживань, навіть якщо механізми захисту були обійдені

3. Обмеження зменшується розмір утрат, якщо злочин усе-таки відбувся незважаючи на заходи для його запобігання і виявлення.

4. Відновлення - забезпечується ефективне відновлення інформації при наявності документованих і перевірених планів по відновленню

Перші комп'ютери були доступні тільки невеликому числу людей, що вміли їх використовувати. Вони містилися в спеціальних приміщеннях, вилучених територіально від приміщень, де працювали службовці. Сьогодні комп'ютерні термінали і настільні комп'ютери використовуються скрізь. Число службовців в організації, що мають доступ до комп'ютерного устаткування й інформаційної технології, постійно росте. Чим більше людей одержувало доступ до інформаційної технології і комп'ютерного устаткування, тим більше виникало можливостей для здійснення комп'ютерних злочинів.

З соціальної сторони комп'ютерним злочинцем може бути:

- кінцевий користувач, не технічний службовець і не хакер;
- той, хто не знаходиться на керівній посаді;
- розумний, талановитий співробітник;
- той, хто не розбирається в комп'ютерах;

Типовий комп'ютерний злочинець - це службовець, якому дозволений доступ до системи, нетехнічним користувачем якої він є. У США комп'ютерні злочини, зроблені службовцями, складають 70-80 відсотків щорічного збитку, зв'язаного з комп'ютерами. Інші 20 відсотків дають дії нечесних і незадоволених співробітників. І відбуваються вони по наступним причинам:

- особиста чи фінансова вигода;
- розвага ;
- помста ;
- спроба домогтися розташування кого-небудь до себе ;
- самовираження ;
- випадковість ;
- вандалізм ;

Значний збиток, близько 60 відсотків усіх утрат, наносять помилки людей. Запобігання комп'ютерних утрат, як через навмисні злочини, так і через ненавмисні помилки, вимагає знань в області безпеки. Однак опитування, зроблені періодично в США, показують, що саме службовці, що малі знання в області комп'ютерної безпеки, були основною причиною виявлення комп'ютерних злочинів.

2. Інформаційна безпека

Те, що в 60-і роки називалося комп'ютерною безпекою, а в 70-і - безпекою даних, зараз більш правильно іменується інформаційною безпекою. Інформаційною безпекою називають заходи для захисту інформації від неавторизованого доступу, руйнування, модифікації, розкриття і затримок у доступі.

Метою інформаційної безпеки є убезпечити цінності системи, захистити і гарантувати точність і цілісність інформації, і мінімізувати руйнування, що можуть мати місце, якщо інформація буде модифікована чи зруйнована.

Інформаційна безпека вимагає обліку всіх подій, які можуть бути вбудовані в сам комп'ютер. Інші можуть бути вбудовані в програми. Ухвалення рішення про вибір рівня складності технологій для захисту системи вимагає встановлення критичності інформації. Під критичними даними будемо розуміти дані, що вимагають захисту через імовірність нанесення (ризику) збитку і його величини в тому випадку, якщо відбудеться випадкове чи навмисне розкриття, зміна, чи руйнування даних. Звісно П'ЯТЬ ОСНОВНИХ ТЕХНОЛОГІЙ, ЩО ВИКОРИСТОВУВАЛИСЯ ПРИ КОМП'ЮТЕРНИХ ЗЛОЧИНАХ:

1. Шахрайства.
    1. Уведення неавторизованої інформації;
    2. Маніпуляції дозволеної для введення інформацією;
    3. Неправильне використання файлів з інформацією;
    4. Створення неавторизованих файлів з інформацією;
    5. Обхід внутрішніх мір захисту;
2. Зловживання.
    1. Крадіжка комп'ютерного часу, програм, інформації й устаткування;
    2. Створення неавторизованих файлів з інформацією;
    3. Розробка комп'ютерних програм для неслужбового використання ;
    4. Маніпулювання, неправильне використання можливостей по проведенню робіт на комп'ютерах;

З іншої сторони варто розглянути основні методи, що використовувалися для їхнього здійснення. Вони включають:

1. Обман з даними - найпоширеніший метод при здійсненні комп'ютерних злочинів, тому що він не вимагає технічних знань і відносно безпечний. Інформація міняється в процесі її введення в комп'ютер. Наприклад, при введенні документи можуть бути замінені фальшивими, замість робочих дискет підсунуті чужі, і дані можуть бути сфальсифіковані.
2. Сканування. Інформація, що дозволяє одержати доступ до комп'ютерних файлів може бути знайдена після перегляду сміттєвих кошиків. Дискети, flash залишені на столі, можуть бути прочитані, скопійовані, і украдені.
3. “Троянський кінь”. Цей метод припускає, що користувач не помітив, що комп'ютерна програма була змінена таким чином, що містить у собі додаткові

функції. Програма, що виконує корисні функції, пишеться таким чином, що містить сховані функції, що будуть використовувати особливості механізмів захисту системи.

4. “Люк”. Цей метод заснований на використанні схованого програмного чи апаратного механізму, що дозволяє обійти методи захисту в системі. Цей механізм активується неочевидним образом, наприклад, число транзакцій, оброблених у визначений день, викликає запуск неавторизованого механізму.
5. “салямі”. Названий так через те, що злочин відбувається потроху, невеликими частинами, настільки маленькими, що вони непомітні. Звичайно ця технологія супроводжується зміною комп'ютерної програми. Наприклад, платежі можуть округлятися до декількох центів, і різниця між реальною й округленою сумою надходити на спеціально відкритий рахунок зловмисника.

Наступні ознаки можуть свідчити про наявність уразливих місць в інформаційній безпеці:

1. Не розроблено положень про захист інформації чи вони не дотримуються. Не призначений відповідальний за інформаційну безпеку.
2. Паролі пишуться на комп'ютерних терміналах, містяться в загальнодоступні місця, ними поділяються з іншими, чи вони з'являються на комп'ютерному екрані при їхньому введенні
3. Вилучені термінали і комп'ютери зоставляються без догляду в робочі і неробочі години. Дані відображаються на комп'ютерних екранах, залишених без догляду.
4. Не існує обмежень на доступ до інформації, чи на характер її використання. Усі користувачі мають доступ до всієї інформації і можуть використовувати усі функції системи.
5. Не ведеться системних журналів, і не зберігається інформація про тих, хто і для чого використовує комп'ютер.
6. Зміни в програми можуть вноситися без їхнього попереднього затвердження керівництвом.
7. Дозволено робити численні спроби ввійти в систему з неправильними паролями.
8. Дані, що вводяться, не перевіряються на коректність і точність, чи при їхній перевірці багато даних відкидається через помилки в них.
9. Не робиться аналіз інформації, оброблюваної в комп'ютері, з метою визначення необхідного для неї рівня безпеки

Міри захисту інформаційної безпеки:

1. Контролювання доступу як до інформації в комп'ютері, так і до прикладних програм. Ви повинні мати гарантії того, що тільки авторизовані користувачі мають доступ до інформації і додатків.

2. Ідентифікація користувачів. Вимагайте, щоб користувачі виконували процедури входу в комп'ютер, і використовуйте це як засіб для ідентифікації на початку роботи. Щоб ефективно контролювати комп'ютер, може виявитися найбільш вигідним використовувати його як однокористувальницьку систему.

3. Аутентификация користувачів. Використовуйте унікальні паролі для кожного користувача, що не є комбінаціями особистих даних користувачів, для аутентифікації особистості користувача. Упровадите міри захисту при адмініструванні паролів.

Паролі - тільки один з типів ідентифікації - їх знає тільки користувач. Двома іншими типами ідентифікації є щось, чим володіє користувач (наприклад, магнітна карта), чи унікальні характеристики користувача (його голос).

Якщо в комп'ютері мається убудований стандартний пароль (пароль, що убудований у програми і дозволяє обійти заходу для керуванню доступом), обов'язково змініте його. Зробіть так, щоб програми в комп'ютері після входу користувача в систему повідомляли йому час його останнього сеансу і число невдалих спроб установлення

сеансу після цього. Це дозволить зробити користувача складовою частиною системи перевірки журналів.

Правила захисту паролю :

- не поділяєтеся своїм паролем ні з ким;
- вибирайте пароль для угадування важким;
- використовувати рядкові і прописні букви, цифри, чи виберіть знамените виречення. Ще краще дозвольте комп'ютеру самому згенерувати ваш пароль;
- не використовуйте пароль, що є вашою адресою, псевдонімом, ім'ям дружини, телефонним номером чи чим-небудь звісним;
- використовуйте довгі паролі, тому що вони більш безпечні, найкраще від 8 символів;
- забезпечте невідображаємість пароля на екрані комп'ютера при його введенні
- забезпечте відсутність паролів у роздруківках;
- не записуйте паролю на столі, чи стіні терміналі. Тримаєте його в пам'яті;

Адміністрування паролів :

- періодично змінюйте паролі і робіть це не за графіком;
- шифруйте чи робіть що-небудь з файлами паролів, що зберігаються в комп'ютері, для захисту їх від неавторизованого доступу;
- призначайте на посаду адміністратора паролів тільки саму надійну людину;
- змінюйте паролі, коли людина звільняється;
- змушуйте людей розписуватися за одержання паролів;

Розробіть процедури авторизації, що визначають, хто з користувачів повинний мати доступ до тієї чи іншої інформації і додатків. Використовуйте відповідні заходи для упровадженню цих процедур в організації.

Встановіть порядок в організації, при якому для використання комп'ютерних ресурсів, одержання дозволу доступу до інформації, додатків, одержання пароля потрібен дозвіл тих чи інших керівників.

4. Захист файлів. Крім ідентифікації користувачів і процедур авторизації розробіть процедури по обмеженню доступу до файлів з даними:

- обмежте доступ у приміщення, у яких зберігаються файли даних, такі як архіви і бібліотеки даних;
- використовуйте організаційні міри і програмно-апаратні засоби для обмеження доступу до файлів тільки авторизованих користувачів ;

5. Дотримуйтесь обережності у роботі:

- відключайте невикористовуванні термінали;
- закривайте кімнати, де знаходяться термінали;
- розвертайте екрани комп'ютерів так, щоб вони не були видні з боку дверей, вікон і тих місць у приміщеннях, що не контролюються;
- установите спеціальне устаткування, таке як пристрої, що обмежують число невдалих спроб доступу, чи роблящі зворотний дзвоник для перевірки особистості користувачів, що використовують телефони для доступу до комп'ютера;
- програмуйте термінал відключатися після визначеного періоду невикористання;
- виключайте систему в неробочі годинник.

6. Цілісність інформації. Перевіряйте точність інформації за допомогою процедур порівняння результатів обробки з передбачуваними результатами обробки. Наприклад, можна порівнювати суми чи перевіряти послідовні номери. Перевіряйте точність даних, що вводиться, вимагаючи від службовців виконувати перевірки на коректність, такі як:

- перевірки на перебування символів у припустимому діапазоні символів(числовому чи буквеному) ;
- перевірки на перебування числових даних у припустимому діапазоні чисел;
- перевірки на коректність зв'язків з іншими даними, що порівнюють вхідні дані з даними в інших файлах;

- перевірки на розумність, що порівнюють вхідні дані з очікуваними стандартними значеннями;
- обмеження на транзакції, що порівнюють вхідні дані з адміністративно встановленими обмеженнями на конкретні транзакції;
- Робіть перехресні перевірки умісту файлів за допомогою зіставлення числа чи записів контролю суми значень полючи запису.

7. Міри захисту при розробці програм і відповідні політики повинні включати процедури внесення змін у програму, її приймання і тестування до введення в експлуатацію. Політики повинні вимагати :

- дозволу відповідального керівника для внесення змін у програми,
- обмеження списку людей, кому дозволено вносити зміни і явно описувати обов'язки співробітників по веденню документації.

Повинний бути розроблений і підтримуватися каталог прикладних програм.

Повинні матися контрольні журнали для спостереження за тим, хто з користувачів обновляв критичні інформаційні файли.

Якщо критичність інформації, збереженої в комп'ютерах, вимагає контрольних журналів, то важливі як міри фізичного захисту, так і заходи для керування доступом.

У комп'ютерній мережі журнали повинні зберігатися на сервері, а не на робочій станції. Роздруківки контрольних журналів повинні проглядатися досить часто і регулярно.

Рекомендований перелік основних нормативно-правових актів України для використання при провадженні робіт, які здійснюються в межах господарської діяльності в галузі технічного захисту інформації (ТЗІ) та криптографічного захисту інформації (КЗІ)

1. Закон України "Про інформацію"
2. Закон України "Про ліцензування певних видів господарської діяльності"
3. Закон України "Про захист інформації в автоматизованих системах"
4. Закон України "Про державну таємницю"
5. Указ Президента України від 10.04.2000 №582 "Про заходи щодо захисту інформаційних ресурсів держави".
6. Концепція технічного захисту інформації в Україні.
   Затверджено постановою Кабінету Міністрів України від 08.10.97р. №1126.
7. Положення про технічний захист інформації в Україні.
   Затверджено Указом Президента України від 27.09.99р. № 1229.
8. Положення про контроль за функціонуванням системи технічного захисту інформації.
   Затверджено наказом ДСТСЗІ СБ України від 22.12.99.№ 61.
9. Положення про державну експертизу в сфері технічного захисту інформації.
   Затверджено наказом ДСТСЗІ СБ України від 29.12.99.№ 62.
10. Положення про забезпечення режиму секретності під година обробки інформації, що становить державну таємницю, в автоматизованих системах.
    Затверджено постановою Кабінету Міністрів України від 16.02.98р. № 180.
11. Положення про порядок опрацювання, прийняття, перегляду та скасування міжвідомчих нормативних документів системи технічного захисту інформації.
    Затверджено наказом ДСТЗІ від 01.07.96р. № 44.
12. ДСТУ 3396 0-96 Захист інформації. Технічний захист інформації. Основні положення.
    Затверджено наказом Держстандарту України від 11.10.96 р. № 423.
13. ДСТУ 33961-96 Захист інформації. Технічний захист інформації. Порядок проведення робіт.
    Затверджено наказом Держстандарту України від 19.12.96 р. № 511.

14. ДСТУ 3396.2-97 Захист інформації. Технічний захист інформації. Терміни та визначення.
    Затверджено наказом Держстандарту України від 11.04.97р. №200.
15. ДБН А.2.2-2-96 Проектування. Технічний захист інформації. Загальні вимоги до організації проектування та проектної документації для будівництва.
    Затверджені наказом Держкоммістобудування України від 02.09.96р. № 156.
16. Тимчасове положення про категоріювання об'єктів (ТПКО-95).
    Затверджено наказом ДСТЗІ від 10.07.95р. № 35.
17. Ліцензійні умови провадження господарської діяльності, пов'язаної з розробленням, виробництвом, впровадженням, обслуговуванням, дослідженням ефективності систем і засобів технічного захисту інформації.
    Затверджено наказом Державного комітету України з питань регуляторної політики та підприємництва, ДСТСЗІ СБ України від 29.12.2000р. № 89/67.
18. Тимчасові рекомендації з технічного захисту інформації в засобах обчислювальної техніки, автоматизованих системах і мережах від витоку каналами побічних електромагнітних випромінювань і наведень. (ТР ЕОТ-95).
    Затверджені наказом ДСТЗІ від 09.06.95р. № 25.
19. Тимчасові рекомендації з технічного захисту інформації вид витоку каналами побічних електромагнітних випромінювань і наведень. (ТР ТЗІ-ПЕМВН-95).
    Затверджені наказом ДСТЗІ від 09.06.95р. № 25.
20. НД ТЗІ 1.1-001-99 Технічний захист інформації на програмно-керованих АТС загального користування. Основні положення.
    Затверджено наказом ДСТСЗІ СБ України від 28.05.99р. № 26.
21. НД ТЗІ 1.1-002-99 Загальні положення щодо захисту інформації в комп'ютерних системах від несанкціонованого доступу.
    Затверджено наказом ДСТСЗІ СБ України від 28.04.99р. № 22.
22. НД ТЗІ 1.1-003-99 Термінологія в галузі захисту інформації в комп'ютерних системах від несанкціонованого доступу.
    Затверджено наказом ДСТСЗІ СБ України від 28.04.99р. № 22.
23. НД ТЗІ 1.4-001-2000 Типове положення про службу захисту інформації в автоматизованій системі.
    Затверджено наказом ДСТСЗІ СБ України від 04.12.2000 № 53.
24. НД ТЗІ 1.5-001-2000 Радіовиявлювачі. Класифікація. Загальні технічні вимоги.
    Затверджено наказом ДСТСЗІ СБ України від 13.06.2000. № 29.
25. НД ТЗІ 1.6-001-96 Правила побудови, викладення, оформлення та позначення нормативних документів системи ТЗІ.
    Затверджено наказом ДСТЗІ від 26.07.96р. № 51.
26. НД ТЗІ 2.1-001-2001 Створення комплексів технічного захисту інформації. Атестація комплексів. Основні положення.
    Затверджено наказом ДСТЗІ від 09.02.2001р. № 2.
27. НД ТЗІ 2.3-001-2001 Радіовиявлювачі вимірювальні. Методи та засоби випробувань.
    Затверджено наказом ДСТЗІ від 27.02.2001 р. № 5.
28. НД ТЗІ 2.3-002-2001 Технічний захист мовної інформації в симетричних абонентських аналогових телефонних лініях. Засоби пасивного приховування мовної інформації. Нелінійні атенюатори та загороджувальні фільтри. Методика випробувань.
    Затверджено наказом ДСТСЗІ СБ України від 06.04.2001р. №11.
29. НД ТЗІ 2.3-003-2001 Технічний захист мовної інформації в симетричних абонентських аналогових телефонних лініях. Засоби активного приховування мовної інформації. Генератори спеціальних сигналів. Методика випробувань.
    Затверджено наказом ДСТСЗІ СБ України від 06.04.2001р. № 11.

30. НД ТЗІ 2.3-004-2001 Радіовиявлювачі індикаторні. Методи та засоби випробувань. Затверджено наказом ДСТСЗІ СБ України від 09.04.2001р. № 12.
31. НД ТЗІ 2.5-001-99 Технічний захист інформації на програмно-керованих АТС загального користування. Специфікації функціональних послуг захисту. Затверджено наказом ДСТСЗІ СБ України від 28.05.99р. № 26.
32. НД ТЗІ 2.5-002-99 Технічний захист інформації на програмно-керованих АТС загального користування. Специфікації гарантій захисту. Затверджено наказом ДСТСЗІ СБ України від 28.05.99р. № 26.
33. НД ТЗІ 2.5-003-99 Технічний захист інформації на програмно-керованих АТС загального користування. Специфікації довірчих оцінок коректності реалізації захисту.
Затверджено наказом ДСТСЗІ СБ України від 28.05.99р. № 26.
34. НД ТЗІ 2.5-004-99 Критерії оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу.
Затверджено наказом ДСТСЗІ СБ України від 28.04.99р. № 22.
35. НД ТЗІ 2.5-005-99 Класифікація автоматизованих систем і стандартні функціональні профілі захищеності оброблюваної інформації від несанкціонованого доступу.
Затверджено наказом ДСТСЗІ СБ України від 28.04.99р. № 22.
36. НД ТЗІ 2.5-006-99 Класифікатор засобів копіювально-розмножувальної техніки. Затверджено наказом ДСТСЗІ СБ України від 26.07.99р. № 34.
37. НД ТЗІ 2.7-001-99 Технічний захист інформації на програмно-керованих АТС загального користування. Порядок виконання робіт. Затверджено наказом ДСТСЗІ СБ України від 28.05.99р. № 26.
38. НД ТЗІ 2.7-002-99 Методичні вказівки з використання засобів копіювально-розмножувальної техніки.
Затверджено наказом ДСТСЗІ СБ України від 26.07.99р. № 34.
39. НД ТЗІ 3.6-001-2000 Технічний захист інформації. Комп'ютерні системи. Порядок створення, впровадження, супроводження та модернізації засобів технічного захисту інформації від несанкціонованого доступу. Затверджено наказом ДСТСЗІ СБ України від 20.12.2000р. № 60.
40. НД ТЗІ 3.7-001-99 Методичні вказівки щодо розробки технічного завдання на створення комплексної системи захисту інформації в автоматизованій системі. Затверджено наказом ДСТСЗІ СБ України від 28.04.99р. № 22.
41. НД ТЗІ 3.7-002-99 Технічний захист інформації на програмно-керованих АТС загального користування. Методика оцінки захищеності інформації (базова). Затверджено наказом ДСТСЗІ СБ України від 28.05.99р. № 26.
42. НД ТЗІ 4.7-001-2001 Технічний захист мовної інформації в симетричних абонентських аналогових телефонних лініях. Засоби визначення наявності та віддаленості місця контактного підключення засобів технічної розвідки. Рекомендації щодо розроблення методів випробувань. Затверджено наказом ДСТСЗІ СБ України від 06.04.2001р. №11.
43. НД ТЗІ Р-001-2000 Засоби активного захисту мовної інформації з акустичними та віброакустичними джерелами випромінювання. Класифікація та загальні технічні вимоги. Рекомендації.
Затверджено наказом ДСТСЗІ СБ України від 04.09.2000 .№ 41.
44. Положення про порядок здійснення криптографічного захисту інформації в Україні, затверджене Указом Президента України від 22.05.98 № 505.
45. Положення про державний експортний контроль в Україні, затверджене Указом Президента України від 13.02.98 № 117.
46. Декрет Кабінету Міністрів України від 10.05.93 № 46-93 "Про стандартизацію і сертифікацію".

47. Постанова Кабінету Міністрів України від 14.11.2000 № 1698 "Про затвердження переліку органів ліцензування".
48. Постанова Кабінету Міністрів України від 29.10.2000 № 1755 "Про термін дії ліцензії на провадження певних видів господарської діяльності, розміри і порядок зарахування плати за її видачу".
49. Ліцензійні умови провадження господарської діяльності з розроблення, виробництва, використання, експлуатації, сертифікаційних випробувань, тематичних досліджень, експертизи, ввезення, вивезення крипто систем і засобів криптографічного захисту інформації, надання послуг у галузі криптографічного захисту інформації, торгівлі крипто системами і засобами криптографічного захисту інформації, затверджені наказом Державного комітету України з питань регуляторної політики та підприємництва, Департаменту спеціальних телекомунікаційних систем та захисту інформації СБ України від 29.12.2000 №88/66 і зареєстровані в Міністерстві юстиції України 20.01.2001 за №49/5240.
50. Положення про порядок проведення експертизи в галузі експортного контролю, затверджене постановою Кабінету Міністрів України від 15.07.97 № 767.
51. Положення про порядок контролю за експортом, імпортом і транзитом окремих видів виробів, обладнання, матеріалів, програмного забезпечення і технологій, що можуть використовуватися для створення озброєння, військової чи спеціальної техніки, затверджене постановою Кабінету Міністрів України від 22.08.96 № 1005.
52. Положення про порядок надання суб'єктам зовнішньоекономічної діяльності повноважень на право здійснення експорту, імпорту товарів військового призначення та товарів, які містять відомості, що становлять державну таємницю, затверджене постановою Кабінету Міністрів України від 08.06.98 № 838.
53. Інструкція про порядок обліку, зберігання і використання документів, справ, видань та інших матеріальних носіїв інформації, які містять конфіденційну інформацію, що є власністю держави, затверджена постановою Кабінету Міністрів України від 27.11.98 № 1893.
54. Звід відомостей, що становлять державну таємницю України, затверджений наказом Державного комітету України з питань державних секретів від 31.07.95 №47 і зареєстрований у Міністерстві юстиції України 03.08.95 за № 278/814.
55. Інструкція про порядок забезпечення режиму безпеки, що винна бути створений на підприємствах, установах т організаціях, які здійснюють підприємницьку діяльність у галузі криптографічного захисту конфіденційної інформації, що є власністю держави, затверджена наказ Департамент спеціальних телекомунікаційних систем т захисту інформації СБ України від 22.10.99 № 45 і зареєстрована у Міністерстві юстиції України 29.11.99 за № 817/4110.
56. Положення про порядок розроблення, виготовлення та експлуатації засобів криптографічного захисту конфіденційної інформації, затверджене наказом Департаменту спеціальних телекомунікаційних систем та захисту інформації СБ України від ЗО. 11.99 №53 і зареєстроване в Міністерстві юстиції України 15.12.99 за № 868/4161.
57. Тимчасова інструкція про порядок постачання і використання ключів до засобів криптографічного захисту інформації, затверджена спільним наказом Держстандарту України та Служби безпеки України від 28.11.97 № 708/156 і зареєстрована в Міністерстві юстиції України 17.12.97 за №598/2402.
58. Порядок видачі сертифікатів затвердження типу засобів вимірювальної техніки, сертифікатів відповідності засобів вимірювальної техніки затвердженому типу та свідоцтв про визнання затвердження типу засобів вимірювальної техніки, затверджений наказом Держстандарту України від 31.01.97 № 56 і зареєстрований у Міністерстві юстиції України 15.04.97 за № 137/1941.

59. Порядок проведення робіт із сертифікації продукції іноземного виробництва, що виготовляється серійно, затверджений наказом Держстандарту України від 18.08.98 №633 і зареєстрований у Міністерстві юстиції України 14.10.98 за № 657/3097.
60. Правила визначення вартості робіт із сертифікації продукції та послуг, затверджені наказом Держстандарту України від 10.03.99 № 100 і зареєстровані в Міністерстві юстиції України 31.03.99 за № 194/3487.
61. ДСТ 28147-89 "Системи обробки інформації. Захист криптографічна. Алгоритм криптографічного перетворення".
62. ДСТ 34.310-95 "Інформаційна технологія. Криптографічний захист інформації. Процедури вироблення і перевірки електронного цифрового підпису на базі асиметричного криптографічного алгоритму".
63. ДСТ 34.311-95 "Інформаційна технологія. Криптографічний захист інформації. Функція хешування".
64. ДСТУ 1.0-93 Державна система стандартизації України. Основні положення.
65. ДСТУ 1.3-93 Державна система стандартизації України. Порядок розроблення і побудови, викладення та оформлення технічних розумів.
66. ДСТУ 1.4-93 Державна система стандартизації України. Стандарти підприємства. Основні положення.
67. ДСТУ 1.5-93 Державна система стандартизації України. Загальні вимоги до побудови. викладу, оформлення та змісту стандартів.
68. ДСТУ 1.6-97 Державна система стандартизації України. Порядок державної реєстрації галузевих стандартів, стандартів науково-технічних та інженерних товариств і спілок.
69. ДСТУ 2296-93 Національний знак відповідності. Форма, розміри, технічні вимоги та правила застосування.
70. ДСТУ 2462-94 Сертифікація. Основні поняття. Терміни та визначення.
71. ДСТУ 3410-96 Система сертифікації Укрсепро. Основні положення.
72. ДСТУ 3412-96 Система сертифікації Укрсепро. Вимоги до випробувальних лабораторій та порядок їх акредитації.
73. ДСТУ 3413-96 Система сертифікації Укрсепро. Порядок проведення сертифікації продукції.
74. ДСТУ 3417-96 Система сертифікації Укрсепро. Процедура визнання результатів сертифікації продукції, що імпортується.
75. ДСТУ 3419-96 Система сертифікації УкрСЕПРО. Сертифікація систем якості. Порядок проведення.

Повний перелік нормативних-правових документів знаходиться на сайті Департаменту спеціальних телекомунікаційних систем і захисту інформації Служби безпеки України.

3. Авторське право.

З прийняттям 23 грудня 1993 р. Закону України «Про авторське право і суміжні права» програміст перетворився у творчого працівника, а продукт його діяльності став називатися добутком. Якщо до прийняття цього закону програміст мав право одержувати заробітну плату за створення комп'ютерної чи програми бази даних, то тепер він став власником авторського права на свої добутки з усіма наступаючими з цього привілеями.

Привілей перша. За будь-яке використання комп'ютерних програм і баз даних програмісту повинне виплачуватися авторська винагорода незалежна від виплаченої йому заробітної плати за їхнє створення.

Привілей друга. Програміст може передати авторське право на розроблену їм комп'ютерну програму чи базу даних будь-якому особі, не запитуючи дозволу у свого роботодавця, і одержати за це відповідне винагороду.

Привілей третя. Авторські права програміста захищаються державою протягом 50 років після його смерті і передаються його спадкоємцям.

Ще одна з привілей - право на одержання окремої кімнати додаткової житлової площі не менш чим у 20 квадратних метрів, що оплачується на загальних підставах.

Відповідно до Закону України «Про авторське право і суміжні права», авторське право поширюється на добутки науки, літератури і мистецтва. Термін «програмне забезпечення» у Законі не згадується. Дається тільки визначення комп'ютерної програми і бази даних. Причому комп'ютерна програма охороняється як літературний письмовий добуток, а база даних — як збірник добутків.

Будь-яке складне ПО (наприклад, MS Office) чи програми для автоматизації бухгалтерських задач, є об'єктом, що складається з безлічі комп'ютерних програм і баз даних, і воно охороняється як збірник добутків.

Авторське право поширюється на добуток (ПО, комп'ютерну програму і базу даних) незалежно від його призначення і працездатності, а також від того, випущене воно чи не випущено у світ.

Для юридичної охорони добутку необхідно лише, щоб ПО, комп'ютерна чи програма база даних були результатом творчої діяльності і були представлені в об'єктивній формі, що давала б іншим особам можливість відтворювати результат творчості автора. Такою формою є вихідний чи об'єктний код, а також аудіовізуальне відображення, отримані в результаті розробки комп'ютерної програми чи бази даних.

Добуток — це результат творчої діяльності автора, продукт людського мозку. Тому що мозок людини може робити тільки нематеріальні об'єкти, і вони враховуються в бухгалтерії підприємства як нематеріальні активи.

Суб'єкти ринку ПО повинні розрізняти авторське право і право власності на матеріальний об'єкт, на якому записана комп'ютерна чи програма база даних.

Окремі важливі статті закону про авторське право:

1. Стаття 10 Закону України «Про авторське право і суміжні права».

Авторське право і право власності на матеріальний об'єкт, у якому виражений добуток, не залежать друг від друга. Відчуження матеріального об'єкта, у якому виражений добуток, не означає відчуження авторського права і навпаки. Авторське право — це не право на «рукопис» (вихідний чи об'єктовий код), екземпляр дискети, CD-ROM і т. і.) це право, зв'язане з авторством на визначений добуток.

2. Стаття 11 Закону України «Про авторське право і суміжні права». Автор як первинний суб'єкт авторського права.

1. Первинним суб'єктом, якому належить авторське право, є автор.
2. Автором вважається особа, зазначена як автор на екземплярі обнародуваного добутку, на рукописі чи оригіналі твору образотворчого мистецтва, якщо в судовому порядку не буде доведене інше.

Причому, автору необхідно пам'ятати, що відповідно до п. 2 статті 11 Закону України «Про авторське право і суміжні права» автором є не особа, що створила добуток, а особа зазначена на екземплярі обнародуваного добутку. Таке законодавче положення може приводити до плагіату, тобто до випуску добутку під чужим ім'ям. Автору необхідно знати, що всього його права на створений добуток можуть переходити до іншого обличчю тільки за авторським договором.

Користувачі ПО здобувають його по договору, у якому вказується назва комп'ютерної програми (бази даних) і ціна. Вступаючи з продавцем у такі цивільно-правові відносини, покупець здобуває записане на матеріальному об'єкті (дискетах, CD-ROM і інших носіях) ПО для його використання. Висновку авторського договору в даному випадку не потрібно, тому що відтворення одного екземпляра комп'ютерної програми чи бази даних з дискети (CD-ROM) на вінчестер комп'ютера відбувається законним власником екземпляра ПО.

Стаття 18. Закону України «Про авторське право і суміжні права». Вільне відтворення

комп'ютерних програм.

1.Без дозволу чи автора іншої особи, якому належить авторське право на комп'ютерну програму, якщо інше не визначено в договорі, дозволяється здійснювати наступні дії:

- відтворення одного екземпляра комп'ютерної програми,
- адаптацію комп'ютерної програми,

здійснювані особами, що є законними власниками екземпляра комп'ютерної програми для використання комп'ютерної програми на визначеному комп'ютері відповідно до його призначення;

Як правильно захищати авторське право на ПО? Відповідно до Закону України «Про авторське право і суміжні права» будь-яка особа, що володіє авторським правом на комп'ютерну програму чи базу даних, може його зареєструвати в ДААСП. Для реєстрації необхідно оплатити реєстраційний збір і подати заявку установленої форми, до якої повинний бути прикладений один екземпляр вихідних текстів (його фрагментів) комп'ютерної чи програми бази даних. Фрагментами вихідних текстів можуть бути 25 перших і 25 останніх сторінок тексту комп'ютерної чи програми бази даних.

У зв'язку з тим, що будь-яке ПО, призначене для комерційної реалізації, складається з безлічі комп'ютерних програм і баз даних, воно є складним добутком. Реєструючи його як складний добуток, розроблювач (укладач) здобуває юридично оформлене авторство, що дозволяє йому мати авторські права на різні варіанти виконання конкретного ПО, що відрізняється тільки споживчими властивостями. Таким чином, розроблювач може заявити про свої авторські права на серію програмних продуктів. За реєстрацію одного добутку з юридичних осіб стягується реєстраційний збір (234 гр.).

Авторські права на складові частини ПО можна не реєструвати, тому що при правильному оформленні як трудових відносин з авторами ПО, так і цивільно-правових, усі права на розроблювальне ПО будуть зберігатися за роботодавцем. Для цього досить на кожнім екземплярі ПО ставити знак охорони авторських прав, що складається з латинської букви С в колі, імені (найменування) обличчя, що володіє авторським правом, і року першої публікації добутку.

ПЕРЕЛІК суб’єктів господарювання, які мають ліцензії на провадження діяльності в галузі технічного захисту інформації. За станом на 30.10.2002

«IV КАБЕЛЬНІ СИСТЕМИ» Товариство з обмеженою відповідальністю.
«Агентство захисту інформації» Товариство з обмеженою відповідальністю.
«Акорд» Науково-дослідний інститут.
«Безпека» Український центр. Державне підприємство.
«Державний науково-дослідний інститут автоматизованих систем в будівництві»
«Державний міжгалузевий науково-дослідний інститут пріоритетних наукових, технічних та економічних проблем» (НДІ "Вектор").
«Державний центр інформаційної безпеки» Державне підприємство.
«Електрон» Завод телевізійної техніки (дочірнє підприємство ВАТ "Концерн-Електрон").
«Електронмаш» Державне науково-виробниче підприємство.
«Запорізький науково-дослідний інститут радіозв'язку» Державне підприємство.
«Інститут комп`ютерних технологій» Товариство з обмеженою відповідальністю.
«Інститут проблем математичних машин і систем НАН України»
«Інститут програмних систем НАН України»
«Інститут радіофізики та електроніки НАН України»

«Інститут системного аналізу та комп'ютерно-технологічних систем» Дочірнє підприємство Української Академії Наук Національного прогресу.
«Київський політехнічний інститут» Національний технічний університет України.
«Комунар» Виробниче об'єднання Національного космічного агентства України.
«Конструкторське бюро "Дніпровське» Державне закриту акціонерне товариство.
«Львівський науково-дослідний радіотехнічний інститут» Державне підприємство.
«Магніт-Еконт» Дочірнє підприємство Відкритого акціонерного товариства "Електромеханічний завод "Магніт".
«Мотор Січ» Відкрите акціонерне товариство.
«Науково-виробниче підприємство "Орбіта"» Відкрите акціонерне товариство.
«Науково - дослідний інститут електромеханічних приладів» Відкрите акціонерне товариство.
«Науково-дослідний інститут автоматизації промисловості» Відкрите акціонерне товариство.
«Науково-дослідний і проектно-технологічний інститут машинобудування» Відкрите акціонерне товариство.
«Придніпровська залізниця» Державне підприємство.
«Протон» Центральне конструкторське бюро.
«Радіоприлад» Запорізьке державне підприємство.
«Радіокомунікаційні системи та безпека» Науково-виробниче підприємство. Товариство з обмеженою відповідальністю.
«Союз» Науково-дослідний та проектний інститут.
«Сучасні захищені комп'ютерні системи АСКОС» Товариство з обмеженою відповідальністю.
«Східний гірничо-збагачувальний комбінат» Державне підприємство.
«Телекомінвест» Відкрите акціонерне товариство.
«Український Антивірусний Центр» Товариство з обмеженою відповідальністю.
«Український науково-дослідний інститут радіо і телебачення»
«УКРТЕЛЕКОМ» Відкрите акціонерне товариство.
«Укрінформзв'язок» Державне госпрозрахункове підприємство.
«ФАРЛЕП-2000» Акціонерне товариство закритого типу.
«Харківський державний регіональний науково-технічний центр з питань технічного захисту інформації»
«Харківський інститут інформаційних технологій» Товариство з обмеженою відповідальністю.
«Харківський фізико-технічний інститут» Національний науковий центр.
«ХАРТРОН - ІНКОРПОРЕЙТІД» Акціонерне товариство.
«Южспецсервіс» Акціонерне товариство закритого типу.

ПЕРЕЛІК суб'єктів господарювання, які мають ліцензії на провадження діяльності в галузі криптографічного захисту інформації. За станом на 24.11.2002
«А-1906» Військова частина.
«Аваль» Акціонерний Поштово-Пенсійний Банк.
«Акціонерний банк "Андріївський"»
«Акціонерний банк "Експрес-Банк"»
«Державний ощадний банк України» Відкрите акціонерне товариство.
«Державний експортно-імпортний банк України»
«Державний науково-дослідний інститут інформатизації та моделювання економіки»
«Діамант» Акціонерний банк.

«Донецький Акціонерно-комерційний Міський Банк»
«Донкредитінвест» Закриту акціонерне товариство. Комерційний банк.
«Інститут інформаційних технологій» Акціонерне товариство.
«Інститут комп'ютерних технологій» Товариство з обмеженою відповідальністю.
«Інформаційні комп’ютерні системи» Закриту акціонерне товариство.
«Інформсистема» Асоціація.
«Імпульс» Державне підприємство.
«Інформаційні телекомунікаційні системи» Товариство з обмеженою відповідальністю.
«Інфоком» Товариство з обмеженою відповідальністю.
«КОМПАНІЯ "КОНФІДЕНЦІЙНІ ТЕЛЕКОМУНІКАЦІЇ УКРАЇНИ"» Товариство з обмеженою відповідальністю.
«Львівський державний завод "ЛОРТА"» Державне підприємство.
«Львівський науково-дослідний радіотехнічний інститут» Державне підприємство.
«Львівський радіоремонтний завод» Державне підприємство.
«Надра» Акціонерний комерційний банк.
«Науково-дослідний інститут електромеханічних приладів» Відкрите акціонерне товариство.
«Нафтогаз України» Національна акціонерна компанія.
«ПриватБанк » Закриту акціонерне товариство комерційний банк.
«Промзв’язок» Закриту акціонерне товариство.
«Радіокомунікаційні системи та безпека» Науково-виробниче підприємство. Товариство з обмеженою відповідальністю.
«Сітібанк (Україна)» Акціонерний комерційний банк.
«Судовий інформаційний центр» Державне підприємство.
«Укрпошта» Українське державне підприємство поштового зв'язку.
«УкрСиббанк» Акціонерний комерційний інноваційний банк.
«Укртехзв’язок» Закриту акціонерне товариство.

Лекція 2. Критерії оцінки надійних комп'ютерних систем.

Знання критеріїв оцінки інформаційної безпеки вдатно допомогти при виборі і комплектуванні апаратно-програмної конфігурації.

У повсякденній роботі адміністратор безпеки змушений повторювати дії органів, що сертифікують, оскільки система, що обслуговується, час від часу перетерплює зміни і потрібно:

1) оцінювати доцільність модифікацій і їхнього наслідку;

2) відповідним чином коректувати повсякденну практику користування й адміністрування.

Основними джерелами для проектування комп’ютерних систем захисту є:

- жовта книга (TCSEC) США;
- гармонізовані критерії Європейських країн (ITSEC);
- рекомендації X.800;
- концепція захисту інформації комісії при президенті.

Перша робота, називається по кольору обкладинки " Жовтою книгою ", була вперше опублікована у 1983 року. "Жовта книга" пояснює поняття безпечної системи, що "керує, за допомогою відповідних засобів, доступом до інформації, так що тільки належним образом авторизовані особи чи процеси, що діють від їхнього імені, одержують право читати, писати, створювати і видаляти інформацію". Очевидно, що абсолютно безпечних систем не існує. Будь-яку систему можна "зламати", якщо мати у своєму розпорядженні досить великі матеріальні і часові ресурси. Має сенс зміст оцінювати лише ступінь довіри, що розумно зробити тій чи іншій системі.

У "Жовтій книзі" надійна система визначається як "система, що використовує достатні апаратні і програмні засоби, щоб забезпечити одночасну обробку інформації різного ступеня таємності групою користувачів без порушення прав доступу". Ступінь довіри, чи надійність систем, оцінюється по двох основних критеріях:

1). Політика безпеки - набір законів, правил і норм поводження, що визначають, як організація обробляє, захищає і поширює інформацію. Зокрема, правила визначають, у яких випадках користувач має право оперувати з визначеними наборами даних. Чим надійніше система, тим суворіше і повинна бути політика безпеки. У залежності від сформульованої політики можна вибирати конкретні механізми, що забезпечують безпеку системи. Політика безпеки — це активний компонент захисту, що включає в себе аналіз можливих погроз і вибір мір протидії.

2). Гарантованість - міра довіри до архітектури і реалізації системи. Гарантованність може виникати з тестування. Гарантованість показує, наскільки коректні механізми, що відповідають за проведення в життя політики безпеки. Гарантованість можна вважати пасивним компонентом захисту.

Поряд з цим важливим засобом забезпечення безпеки є механізм підзвітності (протоколювання). Надійна система повинна фіксувати всі події, що стосуються безпеки. Ведення протоколів повинне доповнюватися аудитом, тобто аналізом реєстраційної інформації.

Основне призначення надійної обчислювальної бази — виконувати функції монітора звертань, тобто контролювати допустимість виконання суб'єктами визначених операцій над об'єктами. Від монітора звертань потрібно виконання трьох властивостей:

- Ізольованість. Монітор повинний бути захищений від відстеження своєї роботи;
- Повнота. Монітор повинний викликатися при кожнім звертанні, не повинне бути засобів його обходу;
- Верифікуемість. Монітор повинний бути компактним, щоб його можна було проаналізувати і протестувати, будучи упевненим у повноті тестування.

Реалізація монітора звертань називається ядром безпеки. Ядро безпеки — це основа, на якій будуються всі захисні механізми. Крім перерахованих вище властивостей монітора звертань, ядро повинне гарантувати власну незмінність.

Границю надійної обчислювальної бази називають периметром безпеки. Як уже вказувалося, від компонентів, що лежать поза периметром безпеки, узагалі говорячи, не потрібно надійності. З розвитком розподілених систем поняттю "периметр безпеки" додають інший зміст границі володінь визначеної організації. Те, що усередині володінь, вважається надійним, а те, що поза — ні. Зв'язок між внутрішнім і зовнішнім світами здійснюють за допомогою шлюзової системи, що по ідеї здатна протистояти потенційно ненадійному чи навіть ворожому оточенню.

2. Основні елементи політики безпеки

Згідно "Жовтій книзі", політика безпеки повинна містити в собі наступні елементи:

- Довільне керування доступом;.
- Безпечне повторне використання об'єктів;
- Мітки безпеки;
- Примусове керування доступом.

Докладно розглянемо перераховані елементи.

Довільне керування доступом — це метод обмеження доступу до об'єктів, заснований на обліку особистості чи суб'єкта групи, у яку суб'єкт входить. Довільність керування полягає в тому, що деяка особа може за своїм розсудом давати іншим суб'єктам чи відбирати в них права доступу до об'єкта.

Поточний стан прав доступу при довільному керуванні описується матрицею, у рядках якої перераховані суб'єкти, а в стовпцях — об'єкти. У клітках, розташованих на перетинанні рядків і стовпців, записуються засоби доступу, припустимі для суб'єкта стосовно об'єкта. Наприклад, читання, запис, виконання, можливість передачі прав іншим суб'єктам і т. і.

Більшість операційних систем і систем керування базами даних реалізують саме довільне керування доступом. Головна його перевага — гнучкість, головні недоліки — розподілення керування і складність централізованого контролю, а також відірваність прав доступу від даних, що дозволяє копіювати секретну інформацію в загальнодоступні файли.

Безпека повторного використання об'єктів — важливе на практиці доповнення засобів керування доступом, що охороняє від випадкового чи навмисного витягу секретної інформації з "сміття". Безпека повторного використання повинна гарантуватися для областей оперативної пам'яті (зокрема, для буферів з образами екрана, розшифрованими паролями і т. і.), для дискових блоків і магнітних носіїв у цілому.

Сучасні інтелектуальні периферійні пристрої ускладнюють забезпечення безпеки повторного використання об'єктів. Дійсно, принтер може буферезовати кілька сторінок документа, що залишаться в пам'яті навіть після закінчення печатки. Необхідно почати спеціальні міри, щоб "виштовхнути" їх відтіля.

Мітки безпеки потрібні для реалізації примусового керування доступом з суб'єктами й об'єктами. Мітка суб'єкта описує його благонадійність, мітка об'єкта — ступінь закритості інформації, що міститься в ньому.

Згідно "Жовтій книзі", мітки безпеки складаються з двох частин

— рівня таємності

— і списку категорій.

Рівні таємності, підтримувані системою, що може виглядати, наприклад, так:

- абсолютно секретно;
- секретно;

- конфіденційно;
- несекретно.

Для різних систем набір рівнів таємності може розрізнятися.

Категорії утворюють неупорядкований набір. Їхнє призначення — описати предметну область, до якої відносяться дані. У військовому оточенні кожна категорія може відповідати, наприклад, визначеному виду озброєнь. Суб'єкт не може одержати доступ до "чужої" категорії, навіть якщо його рівень благонадійності — "абсолютно таємно". Фахівець з танків не довідається тактико-технічні дані літаків.

Головна проблема, яку необхідно вирішувати в зв'язку з мітками, це забезпечення їхньої цілісності. По-перше, не повинно бути непомічених суб'єктів і об'єктів, інакше в безпеці з'являться легко використовувані дірки. По-друге, при будь-яких операціях з даними мітки повинні залишатися правильними. Особливо це відноситься до експорту й імпорту даних. Наприклад, друкований документ повинний відкриватися заголовком, що містить текстове і/чи графічне представлення мітки безпеки. Аналогічно, при передачі файлу по каналі зв'язку.

Одним із засобів забезпечення цілісності міток безпеки є поділ пристроїв на багаторівневі й однорівняні. На багаторівневих пристроях може зберігатися інформація різного рівня таємності. Однорівневий пристрій можна розглядати як вирождений випадок багаторівневого, коли припустимий діапазон складається з одного рівня. Знаючи рівень пристрою, система може вирішити, чи припустимо записувати на нього інформацію з визначеною міткою. Наприклад, спроба надрукувати зовсім секретну інформацію на принтері загального користування з рівнем "несекретно" зазнає невдачі.

Мітки безпеки, асоційовані із суб'єктами, більш рухливі, чим мітки об'єктів. Суб'єкт може протягом сеансу роботи із системою змінювати свою мітку, не виходячи за визначені для нього рамки. Іншими словами, він може свідомо занижувати свій рівень благонадійності, щоб зменшити імовірність ненавмисної помилки.

Примусове керування доступом засновано на зіставленні міток безпеки суб'єкта й об'єкта. Суб'єкт може читати інформацію з об'єкта, якщо рівень таємності суб'єкта не нижче, ніж в об'єкта, а всі категорії, перераховані в мітці безпеки об'єкта, присутні в мітці суб'єкта. У такому випадку говорять, що мітка суб'єкта домінує над міткою об'єкта. Зміст правила зрозумілий — читати можна тільки те, що дозволене.

Суб'єкт може записувати інформацію в об'єкт, якщо мітка безпеки об'єкта домінує над міткою суб'єкта. Зокрема, "конфіденційний" суб'єкт може писати в секретні файли, але не може — у несекретні. На перший погляд подібне обмеження може показатися дивним, однак воно цілком розумно. Ні при яких операціях рівень таємності інформації не повинний знижуватися.

Описаний засіб керування доступом називається примусовим, оскільки він не залежить від волі суб'єктів (навіть системних адміністраторів). Після того, як зафіксовані мітки безпеки суб'єктів і об'єктів, виявляються зафіксованими і права доступу.

Утім, у реальному житті довільне і примусове керування доступом сполучається в рамках однієї системи, що дозволяє використовувати сильні сторони обох підходів.

Мета підзвітності — у кожен момент часу знати, хто працює в системі і що він робить. Засобу підзвітності поділяються на три категорії:

- Ідентифікація й аутентифікація;
- Надання надійного шляху;
- Аналіз реєстраційної інформації.

Кожен користувач, перш ніж одержати право робити які-небудь дії в системі, повинний ідентифікувати себе. Стандартний засіб перевірки дійсності (аутентифікації) — пароль, хоча в принципі можуть використовуватися також різного роду особисті картки, біометричні пристрої (сканування ока, відбитків пальців) чи їхня комбінація.

Ідентифікація й аутентифікація — перший і найважливіший програмно-технічний рубіж інформаційної безпеки. Якщо не складає проблеми одержати доступ до системи під будь-яким ім'ям, то інші механізми безпеки, наприклад, керування доступом, утрачають зміст. Без ідентифікації користувачів неможливе протоколювання їхніх дій. У силу перерахованих причин перевірці повинно придаватися першорядне значення.

Надійний шлях зв'язує користувача безпосередньо з надійною обчислювальною базою, минаючи інші, потенційно небезпечні компоненти системи. Ціль надання надійного шляху — дати користувачу можливість переконатися в існуванні обслуговуючої його системи. Відносно нескладно реалізувати надійний шлях, якщо використовується не інтелектуальний термінал. Якщо ж користувач спілкується з інтелектуальним терміналом, задача забезпечення надійного шляху стає надзвичайно складною. Як гарантувати, що користувач спілкується зі справжньою програмою login, а не з "Троянським конем"?

Аудит має справу з діями (подіями), так чи інакше торкаються безпека системи. До числа таких подій відносяться:

- Вхід у систему (успішний чи ні);
- Вихід із системи;
- Звернення до вилученої системи;
- Операції з файлами (відкрити, закрити, перейменувати, видалити);
- Зміна привілеїв чи інших атрибутів безпеки (режиму доступу, рівня благонадійності користувача і т.п.).

Якщо фіксувати всі події, обсяг реєстраційної інформації, швидше за все, буде рости занадто швидко, а її ефективний аналіз стане неможливим.

Протоколювання допомагає стежити за користувачами і реконструювати минулі події. Стеження важливе в першу чергу як профілактичний засіб. Можна сподіватися, що багато хто утримаються від порушень безпеки, знаючи, що їхні дії фіксуються.

Реконструкція подій дозволяє проаналізувати випадки порушень, зрозуміти, чому вони стали можливі, оцінити розміри збитку і вжити заходів по недопущенню подібних порушень у майбутньому.

При протоколюванні події записується принаймні наступна інформація:

- Дата і час події;
- Унікальний ідентифікатор користувача — ініціатора дії;
- Тип події;
- Результат дії (успіх / невдача);
- Джерело запиту (наприклад, ім'я термінала);
- Імена порушених об'єктів (наприклад, відкриваються файли, що видаляються,);
- Опис змін, внесених у бази дані захисту (наприклад, нова мітка безпеки об'єкта);
- Мітки безпеки суб'єктів і об'єктів.

Необхідно підкреслити важливість не тільки збору інформації, але і її регулярного і цілеспрямованого аналізу.

Гарантованість — це міра впевненості, з яким можна затверджувати, що для проведення в життя сформульованої політики безпеки обраний придатний набір засобів, і що кожне з цих засобів правильно виконує відведену йому роль.

У "Жовтій книзі" розглядається два види гарантованости — операційна і технологічна. Операційна гарантованість відноситься до архітектурних і реалізаційних аспектів системи, у той час як технологічна — до методів побудови і супроводу.

Операційна гарантованість містить у собі перевірку наступних елементів:

- Архітектура системи.
- Цілісність системи.
- Аналіз таємних каналів передачі інформації.
- Надійне адміністрування.
- Надійне відновлення після збоїв.

Операційна гарантованість — це спосіб переконатися в тім, що архітектура системи і її реалізація дійсно проводять у життя обрану політику безпеки.

Архітектура системи повинна сприяти реалізації дій безпеки, прямо підтримувати їх. Серед архітектурних рішень, що передбачаються "Жовтою книгою":

- Розподіл апаратних і системних функцій по рівнях привілейованості і контроль обміну інформацією між рівнями.
- Захист різних процесів від взаємного впливу за рахунок механізму віртуальної пам'яті.
- Наявність засобів керування доступом.
- Структурованість системи, явне виділення надійної обчислювальної бази, забезпечення компактності цієї бази.
- Виконання принципу мінімізації привілеїв — кожному компоненту дається рівно стільки привілеїв, скільки необхідно для виконання їм своїх функцій.

Цілісність системи означає, що апаратні і програмні компоненти надійної обчислювальної бази працюють належним образом і що мається апаратне і програмне забезпечення для періодичної перевірки цілісності.

Аналіз таємних каналів передачі інформації — тема, специфічна для режимних систем, коли головне — забезпечити конфіденційність інформації. Таємним називається канал передачі інформації, не призначений для звичайного використання.

Розрізняють таємні канали з пам'яттю і тимчасові. Таємні канали з пам'яттю використовують зміни збережених об'єктів. Таємним знаком може бути розмір файлу, ім'я файлу (складене, наприклад, із вхідного імені і паролю суб'єкта), число пробілів між словами і т.д. Таємний канал вважається швидким, якщо з його допомогою можна передавати 100 чи більш біт у секунду.

Тимчасові канали передають інформацію за рахунок зміни тимчасових характеристик процесів — часу обробки запиту, наприклад. Програму, що угадувала задумане число (від 1 до 4). Людині пропонувалося в розумі проробити визначені обчислення, природно, не повідомляючи програмі відповідь. "Гвіздок" програми полягав в тому, що для різних чисел деякі обчислення проробити легко, а для інших — важко. Аналізуючи розподіл часу, витраченого на обчислення, програма угадувала задумане число. Людина, сама того не відаючи, передавала програмі інформацію з таємного тимчасового каналу.

Надійне адміністрування в "Жовтої книзі" означає, що повинні бути логічно виділені три ролі — системного адміністратора, системного оператора й адміністратора безпеки. Фізично ці обов'язки може виконувати одна людина, але, відповідно до принципу мінімізації привілеїв, у кожен момент часу він повинний виконувати тільки одну з трьох ролей. Конкретний набір обов'язків адміністраторів і оператора залежить від специфіки організації.

Надійне відновлення містить у собі два види діяльності — підготовку до збою (відмовленню) і власне відновлення. Підготовка до збою — це і регулярне виконання резервного копіювання, і вироблення планів дій в екстрених випадках, і підтримка запасу резервних компонентів. Відновлення зв'язано з пере завантаженням системи і виконанням ремонтних і/чи адміністративних процедур.

Технологічна гарантованість охоплює весь життєвий цикл системи, тобто періоди проектування, реалізації, тестування, продажі і супроводу. Усі перераховані дії повинні виконуватися відповідно до твердих стандартів, щоб убезпечитися від витоку інформації і нелегальних "закладок".

Перше, на що звичайно звертають увагу, це тестування. Чи виконує постачальник набір тестів, документує його і надає на розгляд атестаційної комісії, що перевіряє повноту набору і виконує свої тести. Узагалі говорячи, тестуванню підлягають як власне механізми безпеки, так і користувальницький інтерфейс до них. Тести повинні показати, що захисні механізми функціонують у відповідності зі своїм описом і що не існує засобів

обходу руйнування захисту. Тести повинні продемонструвати дієвість засобів керування доступом, захищеність реєстраційної й аутентифікаційної інформації.

Національний центр комп'ютерної безпеки США має дві системи для проведення подібних формальних доказів — Gypsy Verification Environment (GVE) компанії Computational Logic, Inc. і Formal Development Methodology (FDM) корпорації UNISYS.

Засоби конфігураційного керування захищають надійну систему в процесі проектування, реалізації і супроводи. Конфігураційне керування містить у собі:

- ідентифікацію, протоколювання й аналіз усіх змін, внесених у надійну обчислювальну базу (незалежно від того, чи йде мова про апаратуру чи програми);
- керування процесом внесення змін.

Конфігураційне керування використовується розробниками програмного забезпечення по розумінням безпеки. Специфіка підходу — у тотальному контролі за змінами і строгій дисципліні їхнього проведення.

Документація — необхідна умова гарантованої надійності системи і, одночасно, інструмент проведення політики безпеки. Згідно "Жовтій книзі", у комплект документації надійної системи повинні входити наступні томи:

- Керівництво користувача по засобах безпеки;
- Керівництво адміністратора по засобах безпеки;
- Тестова документація;
- Опис архітектури.

На практиці потрібно ще принаймні одна книга — письмовий виклад політики безпеки даної організації.

Керівництво користувача по засобах безпеки призначено для звичайних, непривілейованих людей. Воно повинне містити опис про механізми безпеки і засоби їхнього використання. Керівництво повинне давати відповіді принаймні на наступні питання:

- Як входити в систему? Як вводити ім'я і пароль? Як змінювати пароль? Як часто це потрібно робити? Як вибирати новий пароль?
- Як захищати файли й іншу інформацію? Як задавати права доступу до файлів? В яких випадках це потрібно робити?
- Як імпортувати й експортувати інформацію, не порушуючи правил безпеки?

Керівництво адміністратора по засобах безпеки призначено і для системного адміністратора, і для адміністратора безпеки. У керівництві висвітлюються питання початкового конфігурування системи, перелічуються поточні обов'язки адміністратора, аналізується співвідношення між безпекою й ефективністю функціонування.

Типовий зміст керівництва адміністратора містить у собі наступні пункти:

- Які основні захисні механізми?
- Як адмініструвати засоби ідентифікації й аутентифікації? Зокрема, як заводити нових користувачів і видаляти старих?
- Як адмініструвати засоби довільного керування доступом? Як захищати системну інформацію? Як виявляти слабкі місця?
- Як адмініструвати засобу протоколювання й аудита? Як вибирати региструємі події? Як аналізувати результати?
- Як адмініструвати засобу примусового керування доступом? Які рівні таємності і категорії вибрати? Як призначати і змінювати мітки безпеки?
- Як генерувати нову, переконфігуровану надійну обчислювальну базу?
- Як безпечно запускати систему і відновлювати її після збоїв і відмовлень? Як організувати резервне копіювання?
- Як розділити обов'язку системного адміністратора й оператора?

Тестова документація містить описи тестів і їхні результати. Вона повинна містити план тестування й умови, що накладаються на тестове оточення.

Класи безпеки

"Критерії" Міністерства оборони США відкрили шлях до розподілу інформаційних систем по ступеню надійності. У "Жовтій книзі" визначається чотири рівні безпеки (надійності) — D, C, B і A.

Рівень D призначений для систем, визнаних незадовільними. В даний час він містить дві підсистеми керування доступом для ПК. По міру переходу від рівня C до A до надійності систем пред'являються усе більш жорсткі вимоги. Рівні C і B підрозділяються на класи (C1, C2, B1, B2, B3) з поступовим зростанням надійності. Таким чином, усього мається шість класів безпеки — C1, C2, B1, B2, B3, A1. Щоб система в результаті процедури сертифікації могла бути віднесена до деякого класу, її політика безпеки і гарантованість повинні задовольняти вимогам. Оскільки при переході до кожному наступному класу вимоги тільки додаються, ми, будемо виписувати лише те нове, що при суще даному класу, групуючи вимоги в згоді з попереднім викладом.

Отже, нижче випливають критерії оцінки надійних комп'ютерних систем:

- D1 – незадовільна безпека;
- C1, C2 – довільне керування доступом;
- B1, B2, B3 – примусове керування;
- A1 – веріфіцірований захист.

Вимоги до політики безпеки, проведеною системою, відрізняються відповідно до основних напрямків політики, що передбачаються "Жовтою книгою". Розглянемо довільне керування доступом.

Клас C1. Надійна обчислювальна база повинна керувати доступом іменованих користувачів до іменованих об'єктів. Механізм керування (права для власника/групи/інших, списки керування доступом) повинний дозволяти користувачам забезпечувати поділ файлів між індивідами і/чи групами.

Клас C2. На додаток до C1, права доступу повинні гранулювати ся з точністю до користувача. Механізм керування повинний обмежувати поширення прав доступу. Тільки авторизований користувач (наприклад, власник об'єкта) може надавати права доступу іншим користувачам. Всі об'єкти повинні піддаватися контролю доступу. Якщо розглянути повторне використання об'єктів, то видно що клас C2. при виділенні збереженого об'єкта з пулу ресурсів надійної обчислювальної бази необхідно ліквідувати всі сліди попередніх використань.

Гармонізовані критерії європейських країн.

Європейські країни прийняли погоджені критерії оцінки безпеки інформаційних технологій (Information Technology Security Evaluation Criteria, ITSEC) у 1991 року від імені відповідних органів чотирьох країн — Франції, Німеччини, Нідерландів і Великобританії. Користь від використання погоджених критеріїв очевидна для усіх — і для виробників, і для споживачів, і для самих органів сертифікації.

Принципово важливою рисою європейських критеріїв є відсутність апріорних вимог до умов, у яких повинна працювати інформаційна система. В "Критеріях" Міністерства оборони США видна прив'язка до умов урядової системи, що обробляє секретну інформацію. Так, організація, що запитує сертифікаційні послуги, формулює мету оцінки, тобто описує умови, у яких повинна працювати система, можливі погрози її безпеки і надані нею захисні функції. Задача органа сертифікації — оцінити, наскільки повно досягаються поставлені цілі. Таким чином, у термінології "Жовтої книги", Європейські Критерії відносяться до гарантованості безпечної роботи системи. Вимоги до політики безпеки і до наявності захисних механізмів не є складовою частиною Критеріїв.

Класифікація засобів обчислювальної техніки за рівнем захищеності від НСД.

Переходячи до розгляду пропонованої Гостехкомісією класифікації засобів обчислювальної техніки за рівнем захищеності від несанкціонованого доступу до інформації, відзначимо її близькість до класифікації "Жовтої книги".

В ній встановлюється сім класів захищеності засобів обчислювальної техніки (ЗОТ) від НСД до інформації. Найнижчий клас — сьомий, найвищий — перший. Класи підрозділяються на чотири групи, що відрізняються якісним рівнем захисту:

- перша група містить тільки один сьомий клас;
- друга група характеризується дискреційним захистом і містить шостий і п'ятий класи;
- третя група характеризується мандатним захистом і містить четвертий, третій і другий класи;
- четверта група характеризується версифікаційним захистом і містить тільки перший клас.

Приведемо зведену таблицю розподілу показників захищеності по шести класах СВТ.

Таблиця 1. Розподіл показників захищеності по класах СВТ.

Найменування показника	Клас захищеності					
	6	5	4	3	2	1
Дискреційний принцип контролю доступу	+	+	+	=	+	=
Мандатний принцип контролю доступу	-	-	+	=	=	=
Очищення пам'яті	-	+	+	+	=	=
Ізоляція модулів	-	-	+	=	+	=
Маркірування документів	-	-	+	=	=	=
Захист вводу і виводу на відчужений носій інформації	-	-	+	=	=	=
Зіставлення користувача з пристроєм	-	-	+	=	=	=
Ідентифікація й аутентифікація	+	=	+	=	=	=
Гарантії проектування	-	+	+	+	+	+
Реєстрація	-	+	+	+	=	=
Взаємодія користувача з ЗОТ	-	-	-	+	=	=
Надійне відновлення	-	-	-	+	=	=
Цілісність	-	+	+	+	=	=
Контроль модифікації	-	-	-	-	+	=
Контроль дистрибуції	-	-	-	-	+	=
Гарантії архітектури	-	-	-	-	-	+
Тестування	+	+	+	+	+	=
Керівництво користувача	+	=	=	=	=	=
Посібник з КСЗ	+	+	=	+	+	=
Тестова документація	+	+	+	+	+	=
Конструкторська (проектна) документація	+	+	+	+	+	+

Позначення:

“ - ” - немає вимог до даного класу

“ + ” - нові чи додаткові вимоги

“ = ” - вимоги збігаються з вимогами до ЗОТ попереднього класу

Класифікація автоматизованих систем (АС) за рівнем захищеності від НСД

Класифікація автоматизованих систем улаштована інакше. Снову звернемося до відповідного керівному документу:

1. Установлюється дев'ять класів захищеності АС від НСД до інформації.

2. Кожен клас характеризується визначеною мінімальною сукупністю вимог по захисту.
3. Класи підрозділяються на три групи, що відрізняються особливостями обробки інформації в АС.
4. У межах кожної групи дотримується ієрархія вимог по захисту в залежності від цінності (конфіденційності) інформації і, отже, ієрархія класів захищеності АС.
5. Третя група класифікує АС, у яких працює один користувач, допущений до всієї інформації АС, розміщеної на носіях одного рівня конфіденційності. Група містить два класи — 3Б и 3А.
6. Друга група класифікує АС, у яких користувачі мають однакові права доступу (повноваження) до всієї інформації АС, оброблюваної і (чи) збереженої на носіях різного рівня конфіденційності.
7. Група містить два класи — 2Б и 2А.
8. Перша група класифікує багатопользовательскі АС, у яких одночасно обробляється і (чи) зберігається інформація різних рівнів конфіденційності і не всіх користувачів мають право доступу до всієї інформації АС.
9. Група містить п'ять класів — 1Д, 1М, 1В, 1Б и 1А.

Лекція 3. КОМП'ЮТЕРНІ ВІРУСИ, ЇХНІ ВЛАСТИВОСТІ І КЛАСИФІКАЦІЯ

Властивості комп'ютерних вірусів

Персональні комп'ютери, у яких користувач має вільний доступ до всіх ресурсів комп'ютера, відкрили погрозу інформаційної безпеки, що одержала назву комп'ютерного вірусу.

Приклади. Після ретельного аналізу експерти США прийшли до висновку, , що відмовлення системи автоматичного пуску ракет “Пэтриот” (Patriot), розміщених у ході конфлікту в Перській затоці в 1991 році в Туреччині, був обумовлений помилкою в одній із програм комп'ютера пускової установки, що потрапила сюди, як думають, вже в процесі експлуатації. Французькі фахівці установили, що відмовлення викликані комп'ютерним вірусом, що потрапив у бортову систему наведення ракети, найбільше імовірно, на етапі виробництва і зборки. Серед інших версій улучення вірусів у програмне забезпечення не виключають можливість його навмисного уведення фірмою-виробником для одержання в наступному додаткових прибутків за рахунок виконання дорогого ремонту. Були зареєстровані також і інші випадки влучення комп'ютерних вірусів і програмних помилок у бойові системи з більш легкими наслідками.

Як тільки Stuxnet - вырус установлює, що заразив систему, він починає перехоплювати команди системи, що подаються драйверам, змінюючи їхнє функціонування. Як пояснили аналітики Symantec, "Stuxnet міняє вихідну частоту на короткі періоди часу, спочатку задаючи 1410 Гц, потім знижуючи її до 2 Гц, а потім підвищуючи до 1064 Гц... Модифікація вихідної частоти позбавляє автоматичну систему керування можливостей функціонувати належним чином. Процес збагачення урану - типовий приклад такого процесу, де центрифуги повинні обертатися з дуже точною швидкістю протягом тривалих періодів часу для того, щоб виділити уран потрібної чистоти. Якщо ж ці центрифуги перестають обертатися з потрібною високою швидкістю, то тоді процес відділення більше важких ізотопів урану порушується.

Вірус - програма, що володіє здатністю до само відтворення. Така здатність є єдиним засобом, властивим усім типам вірусів. Не тільки віруси здатні до само відтворення. Будь-яка операційна система здатні створювати власні копії. Копії же вірусу не тільки не зобов'язані цілком збігатися з оригіналом, Але і можуть узагалі з ним не збігатися!

Дане визначення найбільш точно відбиває суть справи, однак слід зазначити, що існують досить небезпечні комп'ютерні програми, що не здатні розмножуватися. До «шкідливих програм», крім вірусів, відносяться:

- троянські коні (логічні бомби) ;
- хакерські утиліти схованого адміністрування видалених комп'ютерів («backdoor») ;
- програми, що крадуть паролі доступу до ресурсів Інтернет і інша конфіденційна інформація ;
- «intended»-віруси;
- конструктори вірусів;
- полиморфні-генератори.

1. Троянські коні

До троянських коней відносяться програми, що наносять які - не будь руйнівні дії в залежності від яких-небудь умов при кожнім запуску нищать інформацію на дисках і т.і.

Більшість троянських коней є програмами, що «підробляються» під які-небудь корисні програми, нові версії популярних утиліти чи доповнення до них. Дуже часто вони розсилаються по електронних конференціях. У порівнянні з вірусами «троянські коні» не одержують широкого поширення. Вони або знищують себе разом з іншими даними на диску, або демаскують своя присутність і знищуються постраждалим користувачем.

До «троянських коней» також можна віднести «дропперы» вірусів - заражені файли, код яких підправлений таким чином, що відомі версії антивірусів не визначають вірусу у файлі. Наприклад, файл шифрується яким-небудь спеціальним чи образом упаковується рідко використовуваним архіватором, що не дозволяє антивірусу «побачити» зараження.

Слід зазначити також «злі жарти» (hoax). До них відносяться програми, що не заподіюють комп'ютеру якої - не будь прямої шкоди, однак виводять повідомлення про те, що така шкода вже заподіяна або буде заподіяна при яких - не будь умовах, або попереджають користувача про неіснуючу небезпеку.

До такої ж категорії «злих жартів» можна віднести також свідомо помилкові повідомлення про нові супервіруси. Такі повідомлення періодично з'являються в електронних конференціях і звичайно викликають паніку серед користувачів.

2. **Утиліти схованого адміністрування (backdoor).**

Троянські коні цього класу по своїй суті є досить могутніми утилітами вилученого адміністрування комп'ютерів у мережі. По своїй функціональності вони багато в чому нагадують різні системи адміністрування, розроблювальні і розповсюджувані різними фірмами-виробниками програмних продуктів.

Єдина особливість цих програм змушує класифікувати їх як шкідливі троянські програми: відсутність попередження про інсталяцію і запуск. При запуску троян установлює себе в системі і потім стежить за нею, при цьому користувачу не видається ніяких повідомлень про дії троянця в системі. Більш того, посилання на троянця може не бути у списку активних додатків. У результаті «користувач» цієї троянської програми може і не знати про її присутність у системі, у той час як його комп'ютер відкритий для вилученого керування.

Будучі встановленими на комп'ютер, утиліти схованого керування дозволяють робити з комп'ютером усе, що в них заклав їхній автор: приймати/відсилати файли, запускати і знищувати їх, виводити повідомлення, записувати інформацію, перезавантажувати комп'ютер і т.д.

3. Intended-віруси

До таких вірусів відносяться програми, що на перший погляд є стовідсотковими вірусами, але не здатні розмножуватися через помилки. Наприклад, вірус, що при зараженні або «забуває» помістити в початок файлів команду передачі керування на код вірусу, або записує в неї невірна адреса свого коду, або неправильно встановлює адреса перехоплення переривання і т.і. До категорії «intended» також відносяться віруси, що за приведеними вище причинами розмножуються тільки один раз - з «авторської» копії.

Заразивши який-небудь файл, вони втрачають здатність до подальшого розмноження.

4. **Конструктори вірусів**

Конструктор вірусів - це утиліта, призначена для виготовлення нових комп'ютерних вірусів. Відомі конструктори вірусів для DOS, Windows і макро-вірусів. Вони дозволяють генерувати вихідні тексти вірусів (ASM-файли), об'єктні модулі і/чи безпосередньо заражені файли.

Деякі конструктори (VLC, NRLG) мають стандартний віконний інтерфейс, де за допомогою системи меню можна вибрати тип вірусу, що уражаються об'єкти (COM чи EXE), наявність відсутність само шифровки, протидія відладнику, внутрішні текстові рядки, вибрати ефекти, що супроводжують роботу вірусу, і т.і. Інші конструктори (PS-MPC, G2) не мають інтерфейсу і зчитують інформацію про тип вірусу з конфігураційного файлу.

5. **Мережні віруси**

Програми - мережні віруси використовують з'єднання мережею для поширення від однієї ЕОМ до іншої, тому атакам мережних хробаків піддаються ті ЕОМ, що зв'язані за допомогою каналів зв'язку. Ставши активним в ЕОМ, мережний хробак може поводитися як комп'ютерний вірус. Він може почати вставляти «Троянських коней», чи робити які-небудь руйнівні чи деструктивні дії. У деякому змісті мережні віруси аналогічні комп'ютерним вірусам, тому що мають здатність заражати інші ЕОМ, а також інші програми. Для розмноження мережні віруси використовують мережні засоби, у залежності від типу мережі й ЕОМ. Прикладами мережних засобів можуть:

а) електронна пошта,

б) вилучений запуск задач, за допомогою якого вірус може запустити свою копію в інший ЕОМ,

в) вилучений термінал, за допомогою якого вірус може ввійти в іншу ЕОМ як користувач, а потім видати команди для свого копіювання з однієї ЕОМ в іншу. Потім нова копія мережного хробака запускається на вилученої ЕОМ. У залежності від розміру мережі, мережний вірус поширюється по великому числу ЕОМ за короткий час, тому руйнування, що він може викликати на одній ЕОМ, потрібно множити на число ЕОМ.

Класифікація вірусів

В даний час Касперскому відомо більш 61000 програмних вірусів, їх можна класифікувати по наступним ознаках:

- середовищу;
- спосіб у зараження середовища;
- впливу;
- особливостям алгоритму;

Усі віруси по їхньому цільовому результаті поділяються на двох категорій:

- віруси публічного ефекту((public domain viruses), що для простоти називають звичайними вірусами;
- спеціальні диверсійні програми.

Більшість з відомих на сьогоднішній день вірусів відносяться до першої категорії. Вони розраховані скоріше на гучний ефект, а не на рішення якоїсь доцільної задачі. Для них не визначена конкретна мета. Ці віруси найменш небезпечні для комп'ютерних

систем і інформаційних масивів, принаймні в даний час , коли гарно відпрацьовані методи боротьби з ними.

Друга категорія представляє із себе ретельно складені цілеспрямовані програми-диверсанти, призначені для виконання однієї чітко визначеної функції, наприклад для одержання грошей по рахунку. На сьогоднішній день мається значна кількість прикладів успішних і провалених спроб одержання грошей за допомогою подібних вірусів.

Різниця між “диверсійними програмами” і звичайними вірусами полягає в наступному.

1. Рівень професіоналізму, на якому вони створені. Для того, щоб скласти диверсійну програму потрібно групи програмістів найвищої кваліфікації, що володіють фундаментальними знаннями в області інформатики й обчислювальної техніки і глибоких спеціальних знань конкретної системи, у яку здійснюється проникнення. Дуже показовий приклад з Робертом Моррисом, якому не вистачило кваліфікації для реалізації свого “геніального” задуму.

2. Звичайні віруси створюються окремими особистостями з честолюбних спонукань як реакція індивідуума на невизнання його суспільством для досягнення гучного публічного ефекту. У той же час диверсійні програми призначені для рішення чітко визначеної задачі в інтересах досягнення якоїсь відчутної, матеріальної вигоди. При цьому творці таких програм прагнуть, по можливості, виключити прояв їхнього впливу.

3. Створення і впровадження звичайних вірусів носить випадковий спорадичний характер і викликається такими факторами як образа, соціальна незадоволеність і т.і. Застосування ж диверсійних програм представляє із себе лише частина комплексу заходів, проведених за єдиним планом, під єдиним керівництвом, і погоджених по місцеві, часу і меті.

У залежності від середовища віруси можна розділити на:

- мережні, поширюються по різних комп'ютерних мережах;
- файлові, упроваджуються головним чином у модулі, що виконуються, тобто у файли, що мають розширення COM і EXE;
- завантажувальні, впроваджуються в завантажувальний сектор диска (Boot-сектор) чи в сектор, що містить програму завантаження системного диска (Master Boot Record) ;
- файлово - завантажувальні, заражають як файли, так і завантажувальні сектори дисків;

По способі зараження віруси поділяються на

- резидентні ;
- нерезиденті.

Резидентний вірус при зараженні (інфікуванні) комп'ютера залишає в оперативній пам'яті свою резидентну частина, що потім перехоплює звертання операційної системи до об'єктів зараження (файлам, завантажувальним секторам дисків і т.і.) і впроваджується в них. Резидентні віруси знаходяться в пам'яті і є активними аж до вимикання пере завантаження комп'ютера.

Нерезиденті віруси не заражають пам'ять комп'ютера і є активними обмежений час.

По **ступені впливу** віруси можна розділити на наступні види:

- *безпечні*, що не заважають роботі комп'ютера, але зменшують обсяг вільної оперативної пам'яті і пам'яті на дисках, дії таких вірусів виявляються в яких-небудь графічних чи звукових ефектах;
- *небезпечні* віруси, що можуть привести до різних порушень у роботі комп'ютера;

- *дуже небезпечні*, вплив яких може привести до втрати програм, знищенню даних, запису інформації в системних областях диска.

По **особливостях алгоритму** віруси важко класифікувати через велику розмаїтість.

Найпростіші віруси - паразитичні, вони змінюють уміст файлів і секторів диска і можуть бути досить легко виявлені і знищені.

Можна відзначити *вірусы-реплікаторы*, що поширюються по комп'ютерних мережах, обчислюють адреси мережних комп'ютерів і записують по цих адресах свої копії.

Відомі *стелс-віруси*, що дуже важко знайти і знешкодити, тому що вони перехоплюють звертання операційної системи до уражених файлів і секторів дисків і підставляють замість свого тіла незаражені ділянки диска.

Найбільше важко знайти *віруси-мутанти*, що містять алгоритми шифровки-розшифровки, завдяки яким копії того самого вірусу не мають ні одного повторюваної ланцюжка байтів.

Маються і так називані *квазівірусні* чи *«троянські»* програми, що хоча і не здатні до само поширення, але дуже небезпечні, тому що, маскуючи під корисну програму, руйнують завантажувальний сектор і файлову систему дисків.

СХЕМИ ФУНКЦІОНУВАННЯ

Повний життєвий цикл комп'ютерного вірусу має наступні етапи:

- упровадження;
- інкубаційний період;
- репродукція (само розмноження) ;
- деструкція, тобто етап, коли “вірус” здійснює руйнівну функцію.

У пам'ять ПЭВМ вірус може бути занесений разом з деякою програмою, у тілі якої він розміщається зловмисником-програмістом. При цьому запуск прикладної програми приводить до запуску вірусу. Моменту активізації програми-вірусу може передувати інкубаційний період, що продовжується від декількох днів до декількох місяців. Це дозволяє сховати джерело проникнення вірусу в комп'ютер. Після закінчення інкубаційного періоду вірус активізується, здійснює репродукцію себе на інші, доступні місця в пам'яті. При цьому вірус може зчіплюватися іншими програмами, впроваджуватися в них. Знаходячись в активному стані програма-вірус може реалізувати деструктивну функцію.

Компонент, що виконує задачу по нанесенню збитку, вступає в дію як правило після завершення роботи тієї частини вірусу, що відповідає за розмноження. Майже завжди для цього компонента чітко визначаються умови початку його роботи (запуску) і передбачається їхня перевірка при кожній активізації вірусу. Якщо ці умови не дотримані, то роботи даного компонента припиняється і не відбувається нічого, щоб могло б упасти в око. Якщо ж зазначені умови дотримуються, то вступає в дію компонентів, що виконує руйнівну функцію. Наслідку при цьому можуть бути різними: від безневинних ефектів на екрані, непояснених порушень роботи портів і до повної утрати всіх даних на твердих чи дисках на дискетах. Досить

часто відбувається дуже складне перекручування даних так, що збиток повною мірою можна знайти тільки по завершенню значного часу.

Завантажувальні віруси

Розглянемо схему функціонування дуже простого завантажувального вірусу, що заражає дискети. Що відбувається, коли ви включаєте комп'ютер? Першою справою керування передається програмі початкового завантаження POST, що зберігається в постійно запам'ятовуючому пристрої (ПЗУ).

Ця програма тестує устаткування і при успішному завершенні перевірок намагається знайти дискету в дисководу А. Серед службових секторів нас поки цікавить один - сектор початкового завантаження (boot-sector). У секторі початкового завантаження зберігається інформація про дискету - кількість поверхонь, кількість доріжок, кількість секторів і ін. Але нас зараз цікавить не ця інформація, а невелика програма початкового завантаження (ПНЗ), що повинна завантажити саму операційну систему і передати їй керування.

Таким чином, нормальна схема початкового завантаження наступна:

POST (ПЗУ) - ПНЗ (диск) - СИСТЕМА

Тепер розглянемо вірус. У завантажувальних вірусах виділяють дві частини - т.зв. голову і т.зв. хвіст. Хвіст, узагалі говорячи, може бути порожнім.

Нехай у вас маються чиста дискета і заражений комп'ютер, під яким ми розуміємо комп'ютер з активним резидентним вірусом. Як тільки цей вірус знайде, що в дисководу з'явилася придатна жертва - у нашому випадку не захищена від запису і ще не заражена дискета, він приступає до зараження. Заражаючи дискету, вірус робить наступні дії:

- виділяє деяку область диска і позначає її як недоступну операційній системі, це можна зробити по-різному, у найпростішому і традиційному випадку зайняті вірусом сектори позначаються як збійні (bad) ;
- копіює у виділену область диска свій хвіст і оригінальний (здоровий) завантажувальний сектор;
- заміщає програму початкового завантаження в завантажувальному секторі своєю головою;
- організує ланцюжок передачі керування відповідно до схеми.

Таким чином, голова вірусу тепер першої одержує керування, вірус встановлюється в пам'ять і передає керування оригінальному завантажувальному сектору. У ланцюжку

POST (ПЗУ) - ПНЗ (диск) - СИСТЕМА

з'являється нова ланка:

POST (ПЗУ) - ВІРУС - ПНЗ (диск) - СИСТЕМА

Мораль ясна: **ніколи не залишайте (випадково) дискет у дисководу А.**

Ми розглянули схему функціонування простого бутового вірусу, що живе в завантажувальних секторах дискет. Як правило, віруси здатні заражати не тільки завантажувальні сектори дискет, але і завантажувальні сектори вінчестерів. Таким чином, на вінчестері маються два об'єкти атаки завантажувальних вірусів - програма початкового завантаження в MBR і програма початкового завантаження в секторі завантажувального диска.

Файлові віруси

Розглянемо тепер схему роботи простого файлового вірусу. На відміну від завантажувальних вірусів, що практично завжди резидентні, файлові віруси зовсім не

обов'язково резидентні. Розглянемо схему функціонування не резидентного файлового вірусу. Нехай у нас мається інфікований файл, що виконується. При запуску такого файлу вірус одержує керування, робить деякі дії і передає керування «хазяїну».

Які ж дії виконує вірус? Він шукає новий об'єкт для зараження - придатний по типі файл, що ще не заражений. Заражаючи файл, вірус впроваджується в його код, щоб одержати керування при запуску цього файлу. Крім своєї основної функції - розмноження, вірус цілком може зробити що-небудь (сказати, запитати, зіграти) - це вже залежить від фантазії автора вірусу. Якщо файловий вірус резидентний,, то він установиться в пам'ять і одержить можливість заражати файли і виявляти інші здібності не тільки під час роботи зараженого файлу. Заражаючи файл, що виконується, вірус завжди змінює його код. Отже, зараження файлу, що виконується, завжди можна знайти. Але, змінюючи код файлу, вірус не обов'язково вносить інші зміни:

◊ він не зобов'язаний змінювати довжину файлу;

◊ не використовувані ділянки коду;

◊ не зобов'язаний змінювати початок файлу;

Нарешті, до файлових вірусів часто відносять віруси, що «мають деяке відношення до файлів», але не зобов'язані впроваджуватися в їхній код. Розглянемо як приклад схему функціонування вірусів відомого сімейства Dir-II. Не можна не визнати, що з'явившись у 1991 р., ці віруси стали причиною дійсної епідемії чуми в Росії. Розглянемо модель, на якій ясно видно основну ідею вірусу. Інформація про файли зберігається в каталогах. Кожен запис каталогу містить у собі ім'я файлу, дату і час створення, деяку додаткову інформацію, номер першого кластера файлу і резервні байти. Останні залишені «про запас» і самої OS не використовуються.

При запуску файлів система зчитує з запису в каталозі перший кластер файлу і далі всі інші кластери. Віруси сімейства Dir-II роблять наступну «реорганізацію» файлової системи: сам вірус записується в деякі вільні сектори диска, що він позначає як збійні. Крім того, він зберігає інформацію про перші кластери файлів, що виконуються, у резервних бітах, а на місце цієї інформації записує посилання на себе.

Таким чином, при запуску будь-якого файлу вірус одержує керування (операційна система запускає його сама), резидентно встановлюється в пам'ять і передає керування викликаному файлу.

Формально, файловими вірусами є і macro-віруси, що заражають файли деяких систем документообігу. В даний час маються віруси для Word for Windows (у величезній кількості!), Word, Excel for Windows. Усі зазначені системи мають убудовані макро-язики (Word Basic, Visual Basic). Ці мови мають достатні можливості, щоб робити практично всі операції, необхідні вірусу. Досить сказати, що маються навіть шифровані і поліморфні macro-віруси. В даний момент більш 90% macro-вірусів – віруси для Word for Windows. Це без сумніву порозумівається тим, що файли цього текстового процесора фактично стали стандартом на текстові документи.

Завантажувально-файлові віруси

Модель завантажувально-файлового вірусу - «популярний» останнім часом завантажувально-файловий вірус OneHalf, що заражає головний завантажувальний сектор (MBR) і файли, що виконуються. Основна руйнівна дія - шифрування секторів вінчестера. При кожнім запуску вірус шифрує чергову порцію секторів, а зашифрувавши половину твердого диска, радісно сповіщає про це. Основна проблема при лікуванні даного вірусу полягає в тому, що недостатньо просто видалити вірус з

MBR і файлів, треба розшифрувати зашифровану їм інформацію. Найбільше «смертельне» дія - проста переписати новий здоровий MBR.

Поліморфні віруси

Поліморфні віруси - віруси, що модифікують свій код у заражених програмах таким чином, що два екземпляри того самого вірусу можуть не збігатися в жодному біті. " Дивлюсь я на небо і думку гадаю ...". Занадто характерний рядок - ми безпомилково довідаємося відомий вірш великого поета. Якщо глянути на питання з погляду вірусолога, то приведений рядок є так називаною сигнатурою – унікальною послідовністю символів, що характеризує вірш. Аналіз вірусів також полягає у виділенні в них сигнатур і наступному їхньому пошуку в потенційних об'єктах вірусної атаки. Досить піймати вірус, вивчити його код, (для професіоналів це справа декількох хвилин) і виділити сигнатуру.

Але віруси не тільки шифрують свій код, використовуючи різні шляхи шифрування, але і містять код генерації шифрувальника і розшифровувача, що відрізняє їх від звичайних шифрувальних вірусів, що також можуть шифрувати ділянки свого коду, але мають при цьому постійний код шифрувальника і розшифровувача.

Поліморфні віруси - це віруси з розшифровувачами, що само модифікуються. Ціль такого шифрування: маючи заражений і оригінальний файли ви все рівно не зможете проаналізувати його код за допомогою звичайного дизассемблювання. Цей код зашифрований і являє собою безглуздий набір команд. Розшифровка виробляється самим вірусом уже безпосередньо під час виконання. При цьому можливі варіанти: він може розшифрувати себе усього відразу, а може виконати таку розшифровку «по ходу справи», може знову шифрувати уже відробили ділянки.

Були придумані алгоритми, що дозволяють не тільки шифрувати код вірусу, але і змінювати розшифровувачі. Цей процес автоматизований — кожна нова копія вірусу містить новий розшифровувач, що може в кожнім біті відрізнятися від розшифровувача її копії, що породила.

Усе це робиться заради утруднення аналізу коду вірусу.

ШЛЯХИ ПРОНИКНЕННЯ ВІРУСІВ У КОМП'ЮТЕР

Основними шляхами проникнення вірусів у комп'ютер є змінні носії (гнучкі і лазерні), а також комп'ютерні мережі. Зараження твердого диска вірусами може відбутися при завантаженні програми з дискети, що містить вірус. Таке зараження може бути і випадковим. Наприклад, якщо дискету не вийняли з дисководу А і перезавантажили комп'ютер, при цьому дискета може бути і не системної. Заразити дискету набагато простіше. На неї вірус може потрапити, навіть якщо дискету просто вставили в дисковод зараженого комп'ютера і прочитали її зміст.

Вірус, як правило, впроваджується в робочу програму таким чином, щоб при її запуску керування спочатку передалося йому і тільки після виконання всіх його команд знову повернулося до робочої програми. Одержавши керування, вірус насамперед переписує сам себе в іншу робочу програму і заражає її. Після запуску програми, що містить вірус, стає можливим зараження інших файлів. Найбільше часто вірусом заражаються завантажувальний сектор диска і файли, що виконуються, що мають розширення EXE, COM, SYS, BAT. Украй рідко заражаються текстові файли.

Після зараження програми вірус може виконати диверсію, не занадто серйозну, щоб не залучити уваги. І нарешті, не забуває повернути керування тій програмі, з якої

був запущений. Кожне виконання зараженої програми переносить вірус у наступну. Таким чином, заразиться все програмне забезпечення.

Для ілюстрації процесу зараження комп'ютерної програми вірусом має сенс уподібнити дискову пам'ять старомодному архіву з папками на тасьмі. У папках розташовані програми, а послідовність операцій по упровадженню вірусу буде в цьому випадку виглядати в такий спосіб.

ОЗНАКИ ПОЯВИ ВІРУСІВ

При зараженні комп'ютера вірусом важливо його знайти. Для цього варто знати про основні ознаки прояву вірусів. До них можна віднести наступні:

- припинення роботи, неправильна робота раніше що успішних програм;
- повільна робота комп'ютера;
- неможливість завантаження операційної системи;
- зникнення файлів і чи каталогів перекручування їхнього вмісту;
- зміна дати і часу модифікації файлів;
- зміна розмірів файлів;
- несподіване значне збільшення кількості файлів на диску;
- істотне зменшення розміру вільної оперативної пам'яті;
- висновок на екран непередбачених чи повідомлень зображень;
- подача непередбачених звукових сигналів;
- часті зависання і збої в роботі комп'ютера;

Слід зазначити, що перераховані вище явища необов'язково викликаються присутністю вірусу, а можуть бути наслідком інших причин. Тому завжди утруднена правильна діагностика стану комп'ютера.

ВИЯВЛЕННЯ ВІРУСІВ І ЗАХОДУ ДЛЯ ЗАХИСТУ І ПРОФІЛАКТИКИ

Як правило, віруси виявляють звичайні користувачі, що зауважують ті чи інші аномалії в поведінці комп'ютера. Вони, у більшості випадків, не здатні самостійно справитися з заразою.

Необхідно лише, щоб якомога швидше вірус потрапив у руки фахівців. Професіонали будуть його вивчати, з'ясовувати, «що він робить», «як він робить», «коли він робить» і ін. У процесі такої роботи збирається вся необхідна інформація про даний вірус, зокрема, виділяється сигнатура вірусу - послідовність байтів, що цілком виразно його характеризує. Для побудови сигнатури звичайно беруться найбільш важливі і характерні ділянки коду вірусу. Одночасно стають ясні механізми роботи вірусу. Наприклад, у випадку завантажувального вірусу важливо знати, де він ховає свій хвіст, де знаходиться оригінальний завантажувальний сектор, а у випадку файлового - спосіб зараження файлу. Отримана інформація дозволяє з'ясувати:

- як знайти вірус, для цього уточнюються методи пошуку сигнатур у потенційних об'єктах вірусної атаки - файлах і \ чи завантажувальних секторах;
- як знешкодити вірус, якщо це можливо, розробляються алгоритми видалення вірусного коду з уражених об'єктів.

Програми виявлення і захисту від вірусів

Для виявлення, видалення і захисти від комп'ютерних вірусів розроблено кілька видів спеціальних програм, що дозволяють виявляти і знищувати віруси. Такі програми називаються *антивірусними*. Розрізняють наступні види антивірусних програм:

- програми-детектори;
- доктори - фаги;
- програми-ревізори;
- програми-фільтри;
- чи вакцини іммунізаторы.

Програми-детектори здійснюють пошук характерної для конкретного вірусу сигнатури в оперативній пам'яті й у файлах і при виявленні видають відповідне повідомлення. Боротись з ним мають або іншій антивірусній програмі, або системний програміст. Недоліком таких антивірусних програм є те, що вони можуть знаходити тільки ті віруси, що відомі розроблювачам таких програм.

Доктора *фаги*, а також *програми-вакцини* не тільки знаходять заражені вірусами файли, але і «лікують» їх, тобто видаляють з файлу тіло програми-вірусу, повертаючи файли у вихідний стан. На початку своєї роботи фаги шукають віруси в оперативній пам'яті, знищуючи їх, і тільки потім переходять до «лікування» файлів. Серед фагов виділяють полифаги, тобто програми-доктора, призначені для пошуку і знищення великої кількості вірусів. Найбільш відомий з них Doctor Web.

З огляду на те, що постійно з'являються нові віруси, програми-детектори і програми-доктори швидко застарівають, і потрібно регулярне відновлення версій.

Програми-ревізори відносяться до самих надійних засобів захисту від вірусів. Ревізори запам'ятовують вихідний стан програм, каталогів і системних областей диска тоді, коли комп'ютер не заражений вірусом, а потім періодично за бажанням користувача порівнюють поточне стан з вихідним. Виявлені зміни виводяться на екран монітора. Як правило, порівняння станів роблять відразу після завантаження операційної системи. При порівнянні перевіряються довжина файлу, код циклічного контролю (контрольна сума файлу), дата і час модифікації, інші параметри. Програми-ревізори мають досить розвиті алгоритми, виявляють стелс-вірусы і можуть навіть очистити зміни версії програми, що перевіряється, від змін, внесених вірусом (Adinf).

Фільтри *«сторожа»* являють собою невеликі резидентні програми, призначені для виявлення підозрілих дій при роботі комп'ютера, характерних для вірусів. Такими діями можуть бути:

- спроби корекції файлів з розширеннями COM, EXE;
- зміна атрибутів файлу;
- прямий запис на диск по абсолютній адресі;
- запис у завантажувальні сектори диска;
- завантаження резидентної програми.

При спробі якої-небудь програми зробити зазначені дії «сторож» посилає користувачу повідомлення і пропонує чи заборонити дозволити відповідну дію. Програми-фільтри дуже корисні, тому що здатні знайти вірус на самій ранній стадії його існування до розмноження. Однак, вони не «лікують» файли і диски. Для знищення вірусів потрібно застосувати інші програми, наприклад фаги. До недоліків програм-сторожів можна віднести їх «настирливість»(наприклад, вони постійно видають попередження про будь-яку спробу копіювання файлу, що виконується,), а також можливі конфлікти з іншим програмним забезпеченням. Прикладом програми-фільтра є програма Vsafe, що входить до складу пакета утиліт MS DOS.

Вакцини *імунізатори* - це резидентні програми, що запобігають зараження файлів. Вакцини застосовують, якщо відсутні програми-доктора, «лікуючі» цей вірус. Вакцинація можлива тільки від відомих вірусів. Вакцина модифікує програму у такий спосіб. Наприклад, записати поле секунд = 62, щоб це не відбивалося на їхній роботі, а вірус буде сприймати їх зараженими і тому не впровадиться. Але антивірус знайде і покалічить. В даний час програми-вакцини мають обмежене застосування.

Апаратні засоби захисту. Маються спеціальні додаткові пристрої, що забезпечують досить надійний захист. На жаль, відомі випадки, коли користувачі, що установили на комп'ютері плату Sheriff, були настільки упевнені у своїй повній невразливості, що зовсім утрачали всяку обережність. Так, вони не звертали уваги на області твердого диска, не захищені Sheriff.

Необхідно пам'ятати, що антивірусні засоби повинні застосовуватися комплексно.

Основні заходи для захисту від вірусів

Для того, щоб не піддати комп'ютер зараженню вірусами і забезпечити надійне збереження інформації на дисках, необхідно дотримувати наступні правила:

- ♦ забезпечте свій комп'ютер сучасними антивірусними програмами і постійно відновляйте їхньої версії;
- ♦ перед зчитуванням з дискет, flash інформації, записаної на інших комп'ютерах, завжди перевіряйте на наявність вірусів, запускаючи антивірусні програми свого комп'ютера;
- ♦ при переносі на свій комп'ютер файлів в архівіруваному виді перевіряйте їх відразу ж після розархівації на твердому диску;
- ♦ періодично перевіряйте на наявність вірусів тверді диски комп'ютера, запускаючи антивірусні програми для тестування файлів, пам'яті і системних областей дисків із захищеної від запису дискети, flash, попередньо завантаживши операційну систему з захищеної від запису системної дискети;
- ♦ завжди захищайте свої дискети, flash від запису при роботі на інших комп'ютерах, якщо на них не буде робитеся запис інформації;
- ♦ обов'язково робіть архівні копії коштовної для вас інформації;
- ♦ не залишайте в кишені дисковода А дискети при включенні, перезавантаж енні операційної системи, щоб виключити зараження комп'ютера завантажувальними вірусами;
- ♦ використовуйте антивірусні програми для вхідного контролю усіх файлів, що виконуються, одержуваних з комп'ютерних мереж.

Лекція 4. Технічні канали і засоби захисту інформації.

Розглянемо перелік технічних каналів в яких потрібно захищати інформацію.

Акустичний канал.

До систем акустичного контролю відносяться різні радіомікрофони, призначенням яких є знімання інформації і передача її по радіоканалі.

Радіомікрофони по своєму виконанню бувають:

- найпростіші - безупинно випромінюючі;
- з включенням на передачу з появою в контрольованому приміщенні розмов чи шумів;
- дистанційно керовані, включаються і виключаються дистанційно на час, необхідне для контролю приміщення.

Спеціальні пристрої знімання інформації і передачі її по радіоканалу можна класифікувати по наступним ознаках:

- діапазону використовуваних частот (від 27 МГц до 1,5 ГГц і вище),
- тривалості роботи (від 5 годин до 1 року);
- радіусу дії (від 15 м до 10 км);
- виду модуляції (АМ, ЧМ, узкополосна ЧМ, однополосна АМ, широко полосна).

Останнім часом з'явилися спеціальні пристрої знімання інформації. До цих каналів можна віднести наступні:

- Пристрою знімання інформації, що ведуть передачу в інфрачервоному. Характеризуються крайньою складністю їхнього виявлення. Термін безупинної роботи - 1-3 доби. Для прийому інформації застосовують спеціальний приймач ИК діапазону, що забезпечує надійний зв'язок на відстані 10-15 м.
- Пристрою знімання інформації, що використовують як канал передачі даних силову електричну мережу 127/220/380 В. Такі пристрої вбудовуються в електричні розетки, подовжувачі, побутову апаратуру й інші місця, де проходить підключається мережа. До основних достоїнств таких пристроїв можна віднести необмежений час роботи. Прийом інформації від таких пристроїв здійснюється спеціальними приймачами, що підключаються до силової мережі, у радіусі до 300 м.
- Пристрою знімання інформації з її закриттям, що використовують шифровку чи перетворення частоти з різними видами модуляції. Спроба прослухати такий пристрій навіть дуже гарним скануючим приймачем ні до чого не приведе - буде чутний лише шум, що вказує тільки на наявність пристрою знімання інформації.
- Пристрою знімання інформації на основі лазерного мікрофона, що дозволяє на відстані до 300 м реєструвати коливання шибок і перетворювати їх у звуковий сигнал.

У тих випадках, коли не можна установити пристрою знімання інформації безпосередньо на об'єкті, застосовують стетоскопні мікрофони , що дозволяють прослухувати переговори через тверду перешкоду (стіну, скло, корпус автомобіля і т.п.). Чим твердіше й однорідніше перешкода, тим краще вони працюють. За допомогою стетоскопного мікрофона можна здійснювати прослуховування розмов через стіну товщиною 1 м і більш.

Телефонні канали зв'язку

Для прослуховування телефонних переговорів використовуються наступні способи підключення:

- паралельне підключення до телефонної лінії. У цьому випадку телефонні радіоретранслятори сутужніше виявляються, але вимагають зовнішнього джерела живлення.
- послідовне включення телефонних радіоретрансляторів у розрив проводу телефонної лінії. У цьому випадку живлення телефонного радіоретранслятору здійснюється від телефонної лінії й в ефір він виходить (починає передачу) з моменту підйому слухавки абонементом.

Способи, якими може вестися прослуховування телефонних ліній і яка при цьому використовується апаратура. Коротко розглянемо ці способи.

1. Безпосереднє підключення до телефонної лінії.

Безпосереднє підключення до телефонної лінії - найбільш простий і надійний спосіб одержання інформації. Підключення телефонного радиоретранслятору може здійснюватися як безпосередньо до телефонного апарата, так і на будь-якій ділянці лінії від телефону абонента до АТС. В даний час існують телефонні радиоретранслятори, що дозволяють прослухувати приміщення через мікрофон лежачої трубки. Для цього на один провід телефонної лінії подають сигнал від генератора високочастотних коливань, а до іншого - підключають амплітудний детектор з підсилювачем. У цьому випадку високочастотні коливання проходять через мікрофон чи елементи телефонного апарата, що володіють "мікрофонним ефектом", і модулюються акустичними сигналами приміщення, які прослухується. У найпростішому випадку застосовується трубка ремонтника-телефоніста, що підключається до лінії в розподільній коробці, де робиться розведення кабелів. Найчастіше, це почерк "фахівців" нижньої ланки криміналу (верхня ланка оснащена апаратурою не гірше державних секретних служб). АТС переключає лінію на розмову при шунтуванні її опором близько 1 ком. Застосування апаратури підслуховування з низькоомним вхідним опором можна досить швидко знайти. Якщо ви почуєте щигликів у лінії чи перепади голосності, - є імовірність того, що вас намагаються прослухати не зовсім професійним способом.

2. *Підкуп персоналу АТС*.

Підкуп обслуговуючого персоналу на АТС - дуже розповсюджений спосіб розкриття ваших секретів. Особливо це стосується міст, де використовуються старі декадно-крокові АТС. Швидше за все, таким способом можуть скористатися злочинні групи або конкуруючі фірми.

3. *Прослуховування через електромагнітний дзвоник.*

Телефонні апарати, де як викличний пристрій використовується електромагнітний дзвоник, поки ще широко поширені в нашій країні. Експерименти показали, що амплітуда ЭДС, що наводиться в лінії, для деяких типів телефонних апаратів може досягати декількох мілівольтів. Корпус апарата є додатковим резонуючим пристроєм

4. *Прослуховування через мікрофон телефонного апарата*

Цей спосіб не є синонімом безпосереднього підключення до лінії. Він набагато складніше. На перший погляд, коли трубка лежить на апараті, немає ніякої можливості використовувати мікрофон як джерело знімання інформаціїДля захисту телефонного апарата від зняття інформації таким способом досить паралельно мікрофону підключити конденсатор ємністю 0,01 - 0,05 мкф. При цьому останній буде шунтувати мікрофон по високій частоті і глибина модуляції ВЧ коливань зменшиться більш ніж у 10 000 разів, що робить подальшу демодуляцію сигналу практично неможливої.

В даний час для збору інформації через оптичні канали можуть використовуватися мініатюрні сховані і спеціальні (камуфліровані під звичайні предмети) фото і відеокамери. Апаратура для схованої фото і відео зйомки, як правило, обладнається спеціальними об'єктивами і насадками:

- мініатюрними об'єктивами, призначеними для зйомки через отвори невеликого діаметра (до 5 мм);
- телескопічними об'єктивами(вирождено телескоп супутника), що дозволяють вести зйомку з далеких відстаней. Такі об'єктиви володіють:

- високою кратністю збільшення (до 1,5 тис. крат);
- комуфляжними об'єктивами, використовуваними для схованої зйомки з різних побутових предметів, наприклад з кейсів;

- об'єктивами, об’єднаними з приладами нічного бачення (з інфрачервоним підсвічуванням), і призначеними для проведення зйомки в темний час доби.

Як приклад устаткування для схованого спостереження може виступати мініатюрна телевізійна камера JT-241s. Сверхминиатюрні об'єктиви дозволяють вести спостереження через отвір діаметром 0,3-1,2 мм. Малі розміри телекамери (39х39х20 мм) дозволяють установити її в будь-які елементи інтер'єра: годинник, книгу, картину, вхідні двері, стіну і т.п.

Канали комп’ютерних мереж

Перехоплення комп'ютерної інформації, несанкціоноване впровадження в бази даних

Стандартність архітектурних принципів побудови устаткування і програмного забезпечення визначають порівняно легкий доступ професіонала до інформації, що знаходиться в персональному комп'ютері (ПК).

Якщо потрапити в приміщення де встановлений комп'ютер, не вдається, у цьому випадку використовують дистанційні способи знімання інформації. Природно, вони ефективні тільки тоді, коли комп'ютер включений. Існують два способи дистанційного зчитування інформації: Перший спосіб заснований на прийомі ВЧ наведень у силову мережу, а другий - на прийомі побічних електромагнітних випромінювань ПК. Поширення побічних електромагнітних випромінювань за межі контрольованої території створює передумови для витоку інформації, тому що можливі її перехоплення за допомогою спеціальних технічних засобів контролю.

У персональному комп'ютері основними джерелами електромагнітних випромінювань є монітор і схеми (пристрою введення і виводу інформації). Причиною могутнього випромінювання є накладення радіосигналу на імпульси розгорнення зображення, вироблювані рядковим трансформатором. Апаратура для цього виду комерційної розвідки досить проста і виготовляється на базі звичайного малогабаритного телевізора. Такі пристрої дозволяють на видаленні 50 м одержувати стійку копію зображення, відображуваного в даний момент на екрані монітора вашого ПК.

Для зменшення рівня побічних електромагнітних випромінювань застосовують спеціальні засоби захисту інформації; екранування приміщень, фільтрацію джерел живленяя, додаткове заземлення, електромагнітне заземлення, а також засоби ослаблення рівнів небажаних електромагнітних випромінювань і наведень за допомогою різних резистивних і поглинаючих погоджених навантажень.

Специфічний канал витоку конфіденційної інформації, що дозволяє несанкціоновано знімати копії (десятки і сотні Гбайт) з інформації, що зберігається на нагромаджувачах на жорстких магнітних дисках ПЭВМ. До особливостей НЖМД, що роблять його привабливим для проведення заходів з використанням ТСР, варто віднести відсутність у країні власного виробництва НЖМД.

Погрози інформації, збереженої на НЖМД, можна умовно підрозділити на:

- комплексні, коли НЖМД зі збереженої на ньому інформацією виступає як елемент програмно-апаратного комплексу;
- прямі, у яких НЖМД виступає як головний і єдиний елемент.

Умови, що забезпечують знімання інформації з НЖМД, можуть бути створені заздалегідь у ході проектування і виготовлення зазначених виробів у виді цільової апаратної чи мікро програмної надмірності, тому що практично всі комп'ютери оснащені імпортними НЖМД. Інший шлях - розміщення додаткових мікро програмних і/чи апаратних засобів фірмами-виготовлювачами комп'ютерів.

Реалізація витоку інформації в обох випадках можлива по наступним узагальненим сценаріям:

- знімання інформації архівіруваної в "технічних" зонах, з НЖМД у процесі його експлуатації в складі ЕОМ чи комп'ютерної мережі.
- нагромадження інформації на НЖМД з наступною імітацією виходу його з ладу.

У першому випадку можлива пряма (з використанням, апаратно-програмних закладок, недокументованих можливостей мережі і т.д.) передача інформації.

Ємкість неформатованого сучасного НЖМД від отформатированого штатними програмними засобами відрізняється на 16-19 %. Це створює можливість поряд з резервуванням інформації на диску під збійні сектори і службову інформацію здійснювати негласне дублювання і збереження конфіденційної інформації. Крім того, вважається, що в зоні для паркувань голівок записи-читання не може міститися ніякої інформації. Для штатних програмних засобів перевірки НЖМД і форматувані високого рівня інформація, що негласно накопичується, буде залишатися невидимої і недоступний.

Другий варіант організації витоку інформації ефективний при цілеспрямованому постачанні комп'ютерної техніки конкретної організації з наступним гарантійним обслуговуванням комп'ютерів у постачальника чи в деякому сервіс-центрі, тому що договір гарантії, як правило, поширюється на все постачання і припускає безоплатну заміну НЖМД при збереженні його товарного виду. У цьому випадку об'єкт сам докладає зусиль по передачі НЖМД із накопиченою інформацією зацікавленій стороні. Фірма, що забезпечує технічне обслуговування, звичайно висуває тверді вимоги по схоронності різного роду маркерів і пломб, що саме по собі гарантує схоронність накопиченої інформації. Після відповідного ремонту інформація, що міститься на НЖМД, стає доступної для копіювання.

Особливо слід зазначити, що при непрацездатності НЖМД він, як правило, відновленню на сервісних станціях на території країни не підлягає і повинний бути переданий на фірму за межі нашої держави. Контроль змісту НЖМД типовими засобами діагностики неможливий, тому що для них він непрацездатний. Стерти інформацію з "несправного" нагромаджувача не можна по тій же причині. А оскільки збережена інформація энергонезалежна, то вона витримує перетинання будь-якої кількості кордонів без якого-небудь митного контролю.

Подальші дії прямо випливають зі ступеня конфіденційності втраченої інформації, збереженої на знешкодженому НЖМД. Якщо цінність інформації велика, то її на НЖМД доцільніше знищити. Знищення інформації може бути досягнуто механічним чи термічним процесами. У першому випадку НЖМД руйнується фізично так, щоб виключити можливість прочитання інформації яким-небудь способом з його робочих дисків. В другому – повну гарантію знищення інформації на накопичувач дає розігрів його до температури 800-1000 градусів Цельсія. У цьому випадку інформація стає абсолютно невідновною. Такий спосіб знищення інформації може бути рекомендований для носіїв, що містять державну таємницю.

Існує і третій (крім механічного і термічних) шлях гарантованого знищення даних. Знищення конфіденційної інформації може бути досягнуте шляхом розміщення НЖМД у досить могутнє постійне магнітне поле, що цілком змінює орієнтацію магнітних доменів на поверхнях НЖМД. У цьому випадку, як і при використанні нагрівання, втрата інформації на НЖМД стає необоротною.

Існують два можливих підходи до створення установок із приведеними вище технічними характеристиками. Перший - необхідний постійний магніт складної форми з концентратором поля. З огляду на технологічні можливості сучасної промислової бази, його створення принципове можливо, але для одиничного екземпляра чи малої серії економічно недоцільно.

Другий підхід припускає використання електромагнітної установки. Однак і тут виникають технологічні проблеми.

Електронні ключі як засіб захисту інформації

Одним з найбільш популярних способів захисту, а саме прив'язки програм до машини, є використання електронного ключа.

Фактично електронний ключ – це спеціалізована мікро ЕОМ, що включає мікропроцесор, пам'ять, а також вузли спеціального призначення – таймери, лічильники, генератори випадкових чисел і т.п.

Існують два основних типи ключів – зовнішні і внутрішні. Внутрішні ключі являють собою плати для ISA і PCI шин, IDE пристрою. Зовнішні підключаються до комунікаційних портів машини – послідовного (COM) і паралельного (LPT) портів. Внутрішні ключі дозволяють реалізувати більш складну і багатофункціональні апаратуру й алгоритми. А основна перевага зовнішніх ключів – підключення, що не потребує розкривати корпусу комп'ютера. Це має важливе значення, коли комп'ютер опечатаний, на гарантійному терміну.

Зовнішньому виконанню характерні і деякі інші особливості, що фактично стали стандартом – прозорість і каскадуемість. Властивість прозорості дозволяє підключати до порту одночасно і ключ, і який-небудь інший пристрій – чи принтер сканер. При цьому ключ не впливає на роботу другого пристрою – ключ прозорий для пристрою. Каскадуемість дозволяє підключати до комп'ютера кілька ключів одночасно, за каскадною схемою. Ця риса властива найчастіше виробам однієї фірми. Головною перевагою такого методу є те, що працездатність захисного механізму не залежить від зміни параметрів BIOS, зміни твердого диска, джамперів материнської плати і т.п. Іншою особливістю є те, що ключ функціонує паралельно і незалежно від CPU, що дозволяє йому виконувати яку-небудь роботу, подібно арифметичному співпроцесору.

Головний з недоліків носить економічний характер – розроблювач програми повинний або сам розробляти ключ, або взяти десь готовий. Користувач, отримуючи захищену програму, повинен придбати і ключ, без якого вона не зможе працювати.

Для захисту від эмуляції протокол обміну з ключем “зашумлюється”, динамічно змінюється і кодується. Деякі моделі мають енергонезалежну пам'ять, доступну для читання/запису з боку комп'ютера.

У більшості випадків ключі – пасивні не вимагають зовнішніх джерел живлення, зберігають записану в них інформацію при відключенні від комп'ютера.

По типу елементної бази сучасні ключі зовнішнього виконання можна розділити на три типи:

1. EEPROM-пам'ять, звичайне ПЗУ зі схемами електронної обв'язки.
2. Замовлені ASIC чипи.
3. Мікропроцесорні ключі.

Розглянемо особливості цих пристроїв.

1. Ключі на базі EEPROM мікросхем. Так побудовані найпростіші і недорогі ключі, орієнтовані на стандартний паралельний порт SPP (Centronics). Каскадування забезпечується мультіплексором, при цьому число одночасно підключених ключів не більш трьох. Такі ключі прозорі тільки для пристроїв, що працюють по SPP протоколу, що викликає проблеми із сучасними лазерними і струйними принтерами типу HP DeskJet, HPLJ-IV, HPLJ-V чи Epson Stylus-850.

Типовий розмір пам'яті таких ключів складає не більш 126 байт. Пам'ять використовується для збереження службової інформації: типу ключа, код користувача, серійний номер ключа і т.п. Деякі ділянки пам'яті можна читати тільки з допомогою спеціального адаптера, що поставляється разом із ключем.

Технологічні обмеження такого підходу (перед записом необхідно виконати попереднє форматування пам'яті обмежує відновлення інформації в ключі і т.д. Нижній поріг напруги живлення таких ключів складає 3-4 вольта, що приводить до нестабільної роботи ключа на сучасних LPT адаптерах. Ключ може самовільно скидатися, перепрограмувати ся або стиратися. Таким чином працездатність багато в чому залежить від якості LPT-порту й умовами програмування самого ключа.

2. Ключі на базі ASIC чипа. Усі функції виконує одна замовлена мікросхема спеціального застосування (ASIC). Ця мікросхема має складну внутрішню організацію і непросту логіку роботи, реалізувати яку, маючи стандартні мікросхеми, неможливо. Такий чип складно відтворити.

При виготовленні ключа для кожного замовника ASIC чистий чип програмується за допомогою спеціальної апаратури (програматора). У нього заноситься унікальний код, привласнений замовнику, а за замовленням і інша інформація. Довжина коду складає 40-50 двійкових розрядів – величезне число варіантів при эмуляції. Деякі ASIC чипи монтують на картах, що вставляються в слот розширення комп'ютера.

Мінімально допустиме напруження живлення 2.0-2.5 в і струм споживання 5/50 мкА в сплячому/робочому режимах. Такі параметри значно знижують вимоги до порту комп'ютера.

Типовою функцією, виконуваної ключем, є необоротна функція y=f(x). Деякі моделі мають таймери і додаткову пам'ять.

3. Ключі на базі мікропроцесора. Призначені для роботи у відкритих системах для захисту UNIX додатків. Такі ключі найчастіше підключаються до послідовного порту RS-232/423 робочої станції і підтримують платформи IBM RS6000, SUN, DEC Alpha і т.д.

Мікропроцесор ключа реалізує складний механізм перетворення даних. Нерідко ключі поставляються чистими, і замовник сам програмує його. Таке рішення збільшує ступінь унікальності ключа. Протокол обміну динамічно змінюється і шифрується. Такі ключі мають і пам'ять ROM і RAM, доступну для користувача.

З вищенаведених даних можна зробити висновок про те, що електронні ключі є найбільш надійним і стійким способом захисту інформації на персональному комп'ютері від несанкціонованого доступу.

Розглянемо приклад організації ключа на базі мікро контролера AT89C52. Він має 4 квазідвунаправлені порти P0-P3. Це дає можливість підключення до нього всіх трьох груп сигналів паралельного порту одночасно. Зробити це можна, наприклад так:

- P0 – 8 ліній даних порту;
- P2 – 5 ліній статусу;
- P3 – 4 лінії керування.

Звідси видно, що порт P1 залишився незадіяним і вільні: 3 розряди порту P2 і 4 розряди порту P3.

Для підключення принтера цього – не вистачає двох ліній. Але їх вистачає для керування якою-небудь периферійною мікросхемою паралельного інтерфейсу, наприклад, КР580ВВ55А. Ця мікросхема має три 8-розрядних вихідних канали, через які на принтер передаються три групи сигналів.

Організувати виходи з відкритим колектором, або звичайним ТТЛ можна, наприклад, з'єднавши разом виходи трьох повторювачів мікросхеми К555ЛП8 і елемента І з відкритим колектором на виході К555ЧИ2 і організувавши керування ними таким чином, щоб у залежності від управляючого сигналу активізувався той чи інший елемент. При використанні такої схеми робота мікро контролера полягає в циклічному прийомі сигналів з комп'ютера і передачі їх через ВВ55А на принтер (і навпаки). Відслідковуються фази реверсування каналу, щоб у потрібні моменти переключити канали передачі даних. Звідси випливає, що в кожному циклі ВВ55А перепрограмується для передачі чергової порції даних. У свою чергу, це ставить вимогу незмінності інформації на шинах даних, керування і статусу.

Основними недоліками розглянутого варіанту є зменшення швидкості передачі даних принтер - комп'ютер у кілька разів і велика імовірність утрати даних. Обидва недоліки можуть бути переборені переносом функцій ретрансляції сигналів з контролера на зовнішні схеми. Це рішення приводить до істотних змін схеми ключа.

Мікросхема К580ВВ55А, що є “вузьким місцем” попередньої схеми (три групи сигналів передаються по черзі по одній 8-розрядній шині) “розпадається” на двунаправлений повторювач ліній даних, 4-розрядний регістр на лініях керування і мультіплексор для ліній статусу. Регістр необхідний для того, щоб контролер міг блокувати принтер при звертанні комп'ютера до ключа, а мультіплексор – для передачі сигналів статусу в комп'ютер як із принтера, так і з контролера.

При такій реалізації контролеру не потрібно відпрацьовувати усі фази кожного протоколу, досить тільки тих, де необхідно переключати напрямок передачі даних. З цього випливає, що для коректної реалізації такого механізму потрібно знати функціонування і тимчасові співвідношення між сигналами для кожного інтерфейсу.

Головною перевагою такої схеми є різке спрощення керуючої програми мікроконтролера, а саме тієї її частини, що забезпечує прозорість для принтера.

Інший варіант реалізації ключа дає оптимальне співвідношення складності схеми і простоти програмування контролера. Однак йому властивий один недолік, зв'язаний з тим, що контролер відслідковує зміну протоколу і його параметрів.

Як було сказано вище, для правильного відстеження цих подій необхідно знати кожен протокол і його тимчасові характеристики. Однак не всі протоколи, що застосовуються в LPT адаптері, жорстко стандартизовані. Цілком стандартизованим є тільки ECP протокол. Negotiation і протоколи SPP, Nibble, Byte стандартизовані, але є програмно эмулюємими, що робить всі їх тимчасові параметри залежними від програмного забезпечення, а саме – від програмних способів формування затримок. Для EPP існують дві версії. Це приведе до того, що ключ, який відслідковує зміни режиму і його параметрів, може або не працювати в деяких режимах, або привести до псування апаратури чи порту принтера. Виходом з такої ситуації є реалізація повторювачів на цілком пасивних схемах, таких як мультиплексори, регістри, повторювачі і т.і. При такій схемній реалізації контролер узагалі не займається спостереженням за зміною інтерфейсів, він тільки очікує специфічної команди входу в захисний механізм.

Застосований у схемі ключа мікро контролер є програмувальним пристроєм. Він має внутрішню Flash-пам'ять, у якій зберігається керуюча програма. Усі дії, виконувані контролером, повинні бути закладені в цю програму.

Програма керування ключа повинна відпрацьовувати наступні фази:

1. Відстеження команди входу в захисний механізм.
2. Реалізація самого захисного модуля.
3. Забезпечення виходу з модуля захисту і повернення в режим чекання.

Захисний механізм найчастіше виконує функції генератора випадкових чисел. Деякі моделі ключів мають автономний таймер, керований із програми, перепрограмувальне ПЗУ й ін.

Відомі програмно-апаратні комплекси.

Акціонерне товариство "Актив", більше відоме в Росії під своєю торговою маркою - NOVEX Software - займається розробкою і реалізацією в Росії і за рубежем систем захисту від комп'ютерного піратства.

NOVEX Software анонсував програмно-апаратний комплекс нового покоління, заснований на електронному ключі NOVEX STEALTH Key. Виконаний на базі мікро

контролера, цей ключ надає найширші можливості по побудові гнучких і одночасно могутніх систем захисту.

Крім захисту NOVEX Key пропонується цілий ряд систем чисто програмного захисту:

- File_PROTECTION - система захисту програм і даних, заснована на створенні ключових дискет.
- Install for Windows - генератор захищених від копіювання дистрибутивів (захист інсталяційного типу для Windows).

Для роботи з електронними ключами серії NOVEX Key здобувається майстер-комплект. Він містить усю математику фірми: мережну, DOS, Windows, а також один універсальний електронний ключ. Майстри-комплекти бувають двох видів: демонстраційний і робочий. Одержавши демонстраційний майстра-комплект, можнае протягом місяця експериментувати з ним, а потім повернути його назад (у цьому випадку Вам буде повернута його вартість) або безкоштовно замінити на робочий. Маючи робочий майстер-комплект, Ви можете здобувати електронні ключі і захищати ними свої програми. Ціна майстра-комплекту NOVEX Key - 25 у.е.

- Будучи користувачем електронних ключів серії NOVEX Key, Ви можете одержати upgrade на NOVEX STEALTH Key безкоштовно.
- Ціна на електронні ключі встановлюється в залежності від кількості ключів,

Стандартний комплекс Акорд має призначення:

- Запобігання доступу до інформації сторонніх облич.
- Виявлення несанкціонованих змін інформації (програмні закладки, віруси та ін.).
- Підвищення безпеки і конфіденційності програмних систем, що використовують електронно-цифровий підпис.

До складу комплексу входить:

1. Плата контролера комплексу "Акорд", встановлювана у вільний слот комп'ютера.
2. Контактний пристрій для, Touch Memory установлюване звичайно на порожній заглушці для дисководів, або в іншім придатному місці.

Touch Memory являють собою мікросхему, розміщену в міцному корпусі по розмірах і формі батарейки від електронного годинника. Кожен прилад має унікальний 48-бітний ідентифікаційний номер, записуваний на кремнієвий кристал лазерним методом. Термін служби Touch Memory не обмежений. Використання таких пристроїв забезпечує переваги:

- висока таємність
- Touch Memory карта містить одну з 248 комбінацій коду, що робить практично неможливим його підбор;
- кожен ідентифікатор володіє унікальним серійним номером (8+48+8біт Rom);
- карта володіє високою механічною міцністю і стійкістю до електромагнітного впливу, витримує температуру від -40 до +850С; гарантійний термін служби більш 10 років;
- миттєве зчитування інформації шляхом легкого торкання пристрою, що зчитує;

Асоціація захисту інформації «Конфідент» пропонує засоби захисту, що використовують Touch Memory як основний засіб аутентифікації.

Сімейство електронних ключів HASP компанії ALADDIN

Ключі виконані на базі могутнього *ASIC-чипа*, що забезпечує як доступ до пам'яті ключа (до 496 байт), так і генерацію серії *y=f(x)*. Цей чип визначає і протокол обміну ключа з портом. У нього ж при виготовленні ключа “прошивається” унікальний код розроблювача (Batch Code). Доступ до пам'яті ключа *індивідуальний* для кожного виробника програм.

Найвища надійність ключа забезпечує до 100,000 циклів перезапису EEPROM-пам'яті і термін збереження даних не менш 10 років.

У захисному механізмі реалізовані могутні антивідладочні й антитрассіровочні алгоритми із шифруванням даних, реалізований захист від вірусів. З одним ключем може

бути захищена довільна кількість додатків. Для дистанційного перепрограмування використовується *RUS* - Remote Update System.

KRYPTON.

Система KRYPTON є найбільш надійною вітчизняною системою. Вона забезпечує захист даних з гарантованою стійкістю і являє собою програмно-апаратний комплекс, призначений для криптографічного захисту даних, розміщених на НЖМД.

Апаратура системи розроблена «АНКАД» і містить у собі плату, на якій розташовані:

- дві 32-розрядні однокристальні мікро ЕОМ, що реалізують державний стандарт шифрування даних з довжиною ключа 256 біт. Обидві мікро ЕОМ працюють у паралельному режимі з контролем результату, що забезпечує високу вірогідність криптографічного перетворення;
- плата ПЗУ, що містить засоби підтримки програмної частини системи KRYPTON, що реалізує і блокування завантаження ПЭВМ із НГМД.

Програмне забезпечення системи забезпечує функції розмежування доступу до комп'ютера і логічних дисків НЖМД. Система здійснює «прозоре» шифрування даних на НЖМД. Швидкість шифрування складає до 100 Кб/с.

Програмне забезпечення завантажується під час завантаження комп'ютера і стає складовою частиною системи BIOS для НЖМД. При цьому забезпечується повне шифрування НЖМД, що запобігає появі «секретного залишку» даних.

Засоби захисту інформації сертифіковані Гостехкоміссиєю Росії

СИСТЕМА ЗАХИСТУ ІНФОРМАЦІЇ ВІД ПДВ, СЕРТИФІКОВАНІ ГОСТЕХКОМИССИЕЙ РОСІЇ	
"Кобра"	Санкт-Петербург,ИМИСС і АОЭТ "Кобра-Лайн"
"Страж-1.1"	Москва, в/ч 01168
"Сніг-1.0"	Москва, ЦНИИАтоминформ
"Dallas Lock 3.1"	Санкт-Петербург, Асоціація "Конфідент"
"Марс"	Москва, Центр "Безпека"
Убудована система парольного захисту завантаження "ПК PCD-4 Gsx/25" фірма Siemens Nixdorf	Москва, Керування інформаційних ресурсів Адміністрації Президента
"Кютак-С"	Санкт-Петербург, ТОО "Факос"
"Снег-ЛВС"	Москва, ЦНИИАтоминформ
"Сизам"	Пенза, НПФ "Кристал"
"Secret NET 1.10"	Москва, АТЗТ "Информзащита"

Для охорони комерційної таємниці наявність сертифікованих засобів необов'язково. Проте технічні фахівці різних приватних структур, особливо банків, що не обробляють інформацію державної важливості (секретну), здебільшого віддають перевагу сертифікованій техніці і програмним продуктам.

Лекція 5. Захист від НСК.

Програми, що не розраховані на масового користувача (системи автоматизованого проектування, спеціалізовані бази даних і т.п.) це наукомісткі товари. Тобто вони мають великий внесок висококласних фахівців у конкретних прикладних областях. Праця таких фахівців високо цінуються, тому загальні витрати на створення подібного роду програм великі. В сполученні з порівняно невеликим колом потенційних користувачів робить ціну кожної копії дуже великою. Такі програми забезпечуються захистом, тому що обмежений попит на них формує дуже специфічний ринок. Покупцями цих програм є фахівці в конкретній прикладній діяльності, що не мають потрібних навичок і бажання займатися «розкриттям» програм.

Будь-яка копія захищеної програми повинна містити в собі чи в зовнішньому файлі «ключ» - одне чи кілька кодових чисел. У момент перевірки програма порівнює специфічні ознаки робітничого середовища з заздалегідь закодованими в ключі і по результату порівняння формує відповідний ознаку. Таким чином, мало створити копію програми. Для того щоб ця копія стала працездатної, їй необхідно передати ключ, набудований на роботу з визначеним комп'ютером.

Такі ключем можуть бути вироблені на підставі аналізу індивідуальних ознак середовища виконання. Якими специфічними ознаками може володіти програмно - апаратне середовище, у якій працює програма? Для ПК цими ознаками можуть бути:

- тип ПК і тип (версія) операційної системи;
- фізичне положення файлів на дисковому носії;
- склад апаратних засобів;
- наявність схованих частин програми;
- фізичні особливості (у тому числі дефекти) носія.

Деякі з цих ознак дуже індивідуальні (наприклад, фізичні особливості неякісного носія), інші мають меншу індивідуальність. Програма може використовувати один чи кілька ознак для перевірки легальності копії. При цьому особливого значення набуває спосіб використання програми:

- якщо програма розрахована на роботу на конкретному ПК, вибираються одні ознаки,
- якщо вона може вільно переміщатися з одного комп'ютера на іншій без утрати працездатності,- інші.

Програми першого типу назвемо стаціонарними, а другого - мобільними. Захист тих і інших програм має свої особливості.

В усіх випадках перевірка легальності не повинна істотно позначатися на швидкодії програми чи очикуванні від користувача якихось додаткових дій. Наприклад, навряд чи можна вважати ефективною систему, яка використовує пароль, що повинен ввести користувач. По-перше, користувач може забути це слово і тим самим позбавитися законно придбаної програми. По-друге, ніщо не заважає йому повідомити це слово будь-якому іншому користувачу. Система захисту не повинна перевіряти користувача, вона повинна перевірити копію! Тому звичайно ці перевірки використовуються в сполученні з перевірками інших індивідуальних ознак і призначені.

Розглянемо можливі атаки на захищену програму і моделі погроз.

- Створення копії ключової дискети.
- Перший крок при проведенні досліджень аналізу прогноз був спрямований на вивчення прийомів роботи хакерів. Було проведене тестування 118 програмістів маючих різну професійну підготовку. Питання тесту і відповіді приведені в табл.1.
-
-
-

- Таблиця1.

Питання	Відповідь (з урахуванням знань архітектури ПЭВМ, мов програмування, стимулу і можливої винагороди). Типи.	
	1	2
Усього тестувалось (людей)	98	20
Розкривати захист не стали	58	6
Спробую скопіювати захищену дискету стандартними засобами	40	14
1) з цікавості	40	14
2) із самоствердження	21	3
3) за винагороду	40	13
4) за солідну винагороду	40	14
Спробую подивитися, як вона улаштована за допомогою стандартних засобів (відладник, дизассемблер)	7	13
1) з цікавості	0	4
2) із самоствердження	0	5
3) за винагороду	3	9
4) за солідну винагороду	7	11
Спробую подумати і застосувати що небудь своє, навіть якщо це своє прийдеться розробити	2	5
1) з цікавості	0	1
2) із самоствердження	0	3
3) за винагороду	0	3
4) за солідну винагороду	2	5
Застосую вже готовий власний спеціальний інструментарій	0	2
1) з цікавості	0	1
2) із самоствердження	0	2
3) за винагороду	0	1
4) за солідну винагороду	0	2

- Типи тестуємих:
  - 1 тип: Знання машинно-незалежних мов програмування й уміння працювати зі стандартними утилітами.
  - 2 тип: Гарне знання асемблера й архітектури ПЭВМ.
  - Що цікаво, такий стимул як самоствердженні не набагато уступає грошовій винагороді, особливо в умовах специфічного ринку. Це і зрозуміло, один із критеріїв професіоналізм розроблювача, - наявність у нього самостійно отриманих листингів із широко відомих механізмів захисту. Тому однієї з важливих характеристик системи захисту є її популярність. Займатися дослідженнями мало відомої системи захисту заради самоствердження ніхто не буде.
- Результати тестування показують, що 98% програмістів не будуть намагатися переборювати захисний механізм, якщо система захисту гарантує:

- - неможливість копіювання носія широко відомими і наявними практично в кожного розроблювача засобами копіювання, такими як комплекс COPY II - PC, COPYWRITE, UNLOCK.
- - неможливість дослідити програмний продукт за допомогою широко відомих відладників, дизассемблерів;
- - відсутність широкої популярності системи захисту.
- Таким чином, можна мати 100% гарантію надійності системи захисту, якщо:
  1. система захисту задовольняє названим 3 основним вимогам;
  2. в організації, куди поставляється захищений продукт немає талановитих програмістів з почуттям не цілком реалізованих здібностей;
  3. відсутня широка популярність у пакета, що захищається;
  4. пакет, що захищається, продається дешевше чим оплата роботи професійного хакера,

Захист від копіювання

З погляду професійного програміста термін "захист від копіювання" для PC, що працює під керуванням MS DOS/Windows, досить умовний, тому що практично завжди можливо переписати інформацію, що знаходиться на твердому диску. Інша справа, що після цього програма може не виконуватися. Таким чином, без санкції фірми-розповсюджувача неможливо одержати працездатний програмний продукт. Тобто, фактично, "захист від копіювання" - це створення засобів, що дають можливість "захисту від несанкціонованого виконання".

Системи захисту від копіювання по виконуваним функціям можна розділити на наступні групи:

- прив’язка до дискети;
- прив’язка до комп’ютеру;
- прив’язка до ключу;
- опитання довідників
- відтер мінування використання ПО

У світовій практиці існують наступні способи поширення програм:

- FreeWare(вільно зі збереженням прав за автором);
- ShareWare(2-4 тижня випробувати, потім чи не використовувати чи оплатити);
- CriptWare(дві версії: демо і зашифрована робоча).

Більшість програм поширюється за принципом AS IS (як є), загальноприйнятим у міжнародній комп'ютерній практиці. Це означає, що за проблеми, що виникають у процесі експлуатації програми, розроблювач і розповсюджувач відповідальності не несуть.

Захист програмного продукту від несанкціонованого копіювання - актуальна задача в зв'язку зі збереженням комерційних і авторських прав фірм і розроблювачів. По зведеннях закордонних фахівців, економічний збиток від "піратського" копіювання програмного забезпечення складає мільярди доларів. Точні втрати установити неможливо через відсутність повних зведень про число "піратських" копій. Вважається,

що з кожної програми робиться від 2 до 15 копій. У Росії 95% використовуваного софта є "піратським".

Однієї з розповсюджених технологій захисту від копіювання, є створення особливо обумовлених дискет. Їхня особливість полягає в тім, що на дискеті створюється спеціально організована мітка, що використовується як ознака її дистрибутивності. Функцію контролю мітки виконує спеціальна частина програми, яка захищається. Після копіювання засобами OS диска, буде скопійована вся інформація, за винятком мітки. При виконанні програми її контролююча частина установить, що диск не дистрибутивний, і перерве виконання програми. Для створення мітки застосовуються програмні й апаратні засоби, а також їхнє комбінування.

Розглянемо прив'язку до дискети, яка має файлову систему FAT12.

Перестановка в нумерації секторів.

При підготовці нової дискети до роботи вона форматується, тобто визначається кількість доріжок, довжина сектора, кількість секторів на доріжці, нумерація секторів і виробляються інші операції. Якщо ці дії здійснюються з установкою параметрів (довжини сектора, нумерації секторів, величини міжсекторного проміжку й ін.), відмінних від прийнятих за замовчуванням для OS, то такий процес будемо називати нестандартним форматуванням дискети.

Один з методів захисту від копіювання ґрунтується на перестановці номерів секторів на доріжці, тобто замість звичайної послідовності 1,2,3,4,5,6,7,8,9 уводиться, наприклад, 1,5,3,7,9,8,6,2,4.

При виконанні програми на скопійованій дискеті її контролююча частина визначає порядок проходження секторів на заданій доріжці. Дискета, на яку здійснене копіювання, була відформатована звичайними засобами, то нумерація секторів установить послідовність.

Надалі контролюючу частину програми здійснює порівняння обчисленого порядку проходження секторів із установленим на дистрибутивній дискеті. І якщо порядок номерів не збігається, те виконання програми припиняється.

Введення однакових номерів секторів на доріжці

Іншою схемою захисту, заснованої на ідеї нестандартного форматування, є спосіб, при якому частина секторів на визначеній доріжці нумерується однаково. Наприклад, 1,2,3,3,3,6,7,8,9. У ці сектори записуються різні дані.

Контролююча частина програми, повинна визначити, чи мається на дискеті кілька секторів з однаковим номером. Для цього вона надсилає запит читання даних із сектора, номер якого повторюється (у нашому прикладі із сектора 3). У цьому випадку контролер НГМД при виводі голівки читання/запису на сектор 3 може вважати будь-як сектор з даним номером. Цей процес повторюється задану кількість разів підряд, і чергові дані порівнюються з отриманими раніше. У випадку їхнього розходження робиться висновок про наявність на дискеті секторів з однаковим номером. Якщо ж за задану кількість повторень циклу N розходження в даних не знайдено, то робиться висновок про одиничність сектора з номером 3.

Якщо дискета, на яку здійснене копіювання, була відформатована звичайними засобами, то нумерація секторів на всіх доріжках установлюється послідовною. Контролююча частина програми, організує перевірку на наявність декількох секторів з однаковими номером. І тому що кожен номер сектора на заданій доріжці присутня тільки один раз, то виконання програми припиняється.

Уведення міжсекторних зв'язків

В одному з методів захисту від копіювання використовується модифікація викладеного вище способу, що застосовує організацію декількох секторів з однаковим номером на заданій доріжці.

Суть методу в наступному. На виділеній доріжці дистрибутивної дискети шляхом застосування спеціальної програми організується кілька секторів з однаковим номером і в них записуються різні дані. Нехай, наприклад, номера на 79 доріжці задані як 2,2,1,4,5,6,7,8,9 і в першому секторі з номером 2 записано "С", а в другому секторі 2 - "В". Береться ще одна доріжка (наприклад, 78), і сектор на ній (наприклад, 1-ий), і в нього записується інформація, "А". У контролюючій частині програми організується читання сектора 1 на 78 доріжці і відразу ж запит читання сектора 2 на 79 доріжці. Тим самим завжди забезпечується перехід до читання інформації з другого сектора з номером 2 на доріжці 79, тобто ми прочитаємо біграмму "АВ" з дистрибутивної дискети.

Нехай було зроблено "піратське" копіювання дискети з програмою на звичайно отформатировану дискету. Контролююча частина програми зчитує 1-ий сектор з 78 доріжки. Далі йде запит читання сектора 2 на 79 доріжці. Тому що сектора на "піратській" дискеті пронумеровані стандартно (тобто 1,2,3,4,5,6,7,8,9), то після запиту сектора 2 буде отримане значення "С". Отже, буде прочитана біграмма "АС".

Таким чином програма установлює, що якщо дискета не дистрибутивна, то прочитується біграмма "АС". Це означає, що сектори 2 з 79 доріжки, що знаходиться на іншій фізичній відстані від початку доріжки в порівнянні з дистрибутивною дискетою, коли зчитується біграмма "АВ". Програма переривається.

Зміна довжини секторів

Ця схема захисту, заснована на методі нестандартного форматування, використовує зміну довжини сектора. Стандартна довжина сектора, з яким працює OS за замовчуванням, - 512 байт. При цьому на доріжці розміщається 18 секторів. У процесі спеціального форматування дискети на заданій доріжці довжина секторів встановлюється або 128, або 256 байт.

Уведена зміна довжини секторів на заданій доріжці визначається контролюючою частиною програми. Це здійснюється шляхом звертання до сектора з максимальним номером, або перевіркою того, що довжина секторів на зазначеній доріжці 128 (чи 256) байт.

Стандартно отформатованна дискета, на яку проведене "піратське" копіювання програми має довжину секторів на всіх доріжках 512 байт (по 9 секторів на доріжці). У ході виконання програми її контролююча частина здійснює перевірку довжини секторів на заданій доріжці. Якщо контролююча частина використовує спосіб звертання до сектора з максимальним номером (більш 18), то такий сектор знайдений не буде і виконання програми припиняється.

Аналогічно працює контролююча частина й у способі, у якому здійснюється перевірка, що довжина секторів 128 (чи 256) байт.

Зміна міжсекторных проміжків.

Таблиця бази дисків, за допомогою якої керується робота НГМД, містить параметр, що визначає величину міжсекторного проміжку. Вибирається доріжка на дискеті, яка спеціальною програмою форматируеться заново зі зміненим значенням міжсекторного проміжку в таблиці бази дисків, що веде до нестандартного розташування секторів.

Контролююча частина програми при звертанні до обраної доріжки робить її аналіз і робить висновок, чи є дискета дистрибутивною.

Використання додаткової доріжки

При цьому способі захисту використовується додаткова (81) доріжка на дистрибутивній дискеті. Помітимо, що на гнучкому диску мається місце для розміщення ще трьох доріжок, понад прийнятих 80. Дисковод дозволяє вести на них запис даних. Тому на додатковій 81-й доріжці на дистрибутивній дискеті зберігається інформація для контролюючої частини програми.

При копіюванні на стандартно отформатовану дискету додатково доріжка сприйнята не буде, що і виявляється контролюючою частиною програми, що захищається.

Ведення логічних дефектів у заданий сектор

Спочатку необхідно розглянути фізичну і логічну будівлю доріжки на гнучкому диску.

	4E					
SYNC	0	1-N	12 bytes			
	A1		Заголовок 1-го сектора, N			
	A1					
	A1					
ID AM	FE					
	C					
	H					
	R					
	N					
	CRC					
	CRC					
GAP2	4E..4E		22 bytes=4E(внутрісекторн.промежуток)			
SYNC	0….0		12 bytes			
DATA AM	A1		Адресний маркер даних 1-го сектора. F8-видалені,FB-нормальні дані.			
	A1					
	A1					
	F8/FB					
	512 data byte of 1 sector		Зона даних 1-го сектора, після форматування = F6			
	CRC		Контрольна сума блоку даних			
	CRC					
GAP3	4E		Міжсекторний проміжок (22-125) байт.			
	…					
	4E					
GAP4B		2,3, ..79	Незаповнений остаток доріжки			

Як видно з рисунку, початок доріжок на дискеті відзначено індексним отвором. Кожен сектор на доріжці має 2 частини: секторний ID маркер і власне дані. ID маркер має шість байтів і додаткові байти, що ідентифікують його для контролера нагромаджувача на гнучких магнітних дисках (НГМД) як секторний маркер, а не як дані. Ці шість байт наступні:

1-ий. Номер доріжки з 0 по 79 включно.

2-ий. Номер чи сторони номер голівки. Верхня сторона позначається 0, нижня - 1.

3-ій. Номер сектора на треку з 1 по 18 включно.

4-ий. Розмір сектора в байтах: 0,1,2 чи 3 (128-,256-,512- чи 1024- байта в секторі). За замовчуванням формат OS має 512 байт у секторі.

5-ий і 6-ий. CRC байти для контролю за допомогою циклічного надлишкового коду можливих помилок.

Чотири перших байти секторного ID маркера позначаються як CHRN. CRC-байти фактично містять 16-бітну контрольну суму CHRN-байт, обчислювану, коли сектор записується. Коли ж сектор читається, контрольна сума пере обчислюється для прочитаних з CHRN даних і порівнюється з даними з байтів CRC. Будь-яка розбіжність викликає CRC-помилку ID маркера.

Наступними за ID маркером йдуть байти маркера початку даних і 512 байт власне даних. Ще два перевірочних байти CRC розташовані за даними. Вони містять контрольну суму 512 байт даних, обчислену при записі сектора. Розбіжність між контрольними сумами, записаними в CRC байтах і обчислених при читанні, визначається як CRC-помилка даних. Після CRC-байтів першого сектора мається міжсекторний проміжок. Потім йдуть інші сектори і міжсекторні проміжки.

Тепер буде зрозуміла ідея методу захисту, заснованого на використанні CRC-помилки даних. Її виникнення досягається в такий спосіб. Стандартна операція запису даних у сектор включає шість кроків:

1. Включення мотора НГМД і коротке чекання його розгону.
2. Виконання операції пошуку заданого сектора і чекання переривання, що вказує на завершення даної операції.
3. Ініціалізація мікросхеми прямого доступу в пам'ять (DMA) 8237 для пересилання даних з пам'яті.
4. Посилка команди запису мікросхеми контролеру НГМД 765 і чекання переривання, що вказує на закінчення пересилання даних.
5. Одержання інформації про статус контролера НГМД.
6. Вимикання мотора.

На дистрибутивній дискеті при виконанні операції запису у визначений сектор під час четвертого кроку викладеної вище процедури виробляється короткочасне відключення мотора НГМД, що приводить до перекручування записуваних даних. Контрольна ж сума даних у CRc байтах буде мати значення, що було отримано по вихідним переданим даним. Так що при читанні даних з цього сектора і підрахунку їхньої контрольної суми отримане значення, буде відрізнятися від запису в CRC-байтах. Таким чином, контрольна частина програми, при спробі читання заданого сектора знайде CRC-помилку даних і визначить дискету як дистрибутивну.

При копіюванні, перекручені дані на заданому секторі сприймаються як правильні і по них підраховується відповідна контрольна сума. Отже, при читанні даного сектора на скопійованій дискеті CRC-помилка даних не виникає, і контролююча частина програми дискету відкидає.

Зміна параметрів дисковода

Однієї зі схем створення захисту від копіювання використовується інша швидкість обертання диска. Стандартна швидкість обертання - 300 об/хв. Якщо її зменшити до 280 хв., у той час як дані для запису передаються з колишньою швидкістю, то це зниження збільшує щільність збереження даних на дискеті. На кожній доріжці утвориться місце для додаткового 10 сектора. Ця обставина є основою захисту, тому що робота з інформацією, записаною на дискеті, яка обертається з новою швидкістю, можлива тільки при відповідній модифікації параметрів дисковода.

Фізичне маркірування дискети

Однієї з найбільш відомих технологій захисту від копіювання є фізична позначка дискети лазерним променем шляхом її пропалювання. Такий дефект приводить до перекручування даних у секторі і виникненню CRC помилки даних при його читанні. При копіюванні позначка не відтворюється на приймаючій дискеті, хоча дані записуються. Таким чином, CRC-помилка даних при читанні сектора з копії не виникає.

Контролююча частина програми роблячи читання відповідного сектора, це встановлює.

Перевірка присутності пробивання може вироблятися контролюючою частиною програми й іншим, більш надійним способом. Вона робить спробу запису в місце, де повинне знаходитися отвір. Невдача - явна ознака того, що дискета дистрибутивна. Якщо ж запис виробляється успішно, то, поверхня в даному місці не ушкоджена і дискета не дистрибутивна.

Розглянута технологія застосовується фірмою Vault Corporation у засобі PROLOK і його подальших модифікаціях.

Застосування фізичного захисного пристрою

Багато апаратно-програмних засобів захисту засновані на тім, що в комп'ютер додається спеціальний фізичний захисний пристрій. При запуску програми її контролююча частина звертається до цього додаткового пристрою, перевіряючи його присутність. Якщо його не знайдено, то виробляється останов програми, або якісь інші дії (наприклад, знищення інформації).

Загальний принцип роботи комп'ютера в цьому випадку наступний. Після запиту на виконання програми відбувається її завантаження в пам'ять і ініціалізація контролюючої частини. На фізичний пристрій захисту, приєднаний до комп'ютера через порт, посилається запит. У відповідь пристрій формує код, що посилається через мікропроцесор в оперативну пам'ять для розпізнавання контролюючою частиною програми. У залежності від правильності коду відповіді програма або переривається, або виконується.

Один з найбільше роз поширених захисних пристроїв це HASP "Алладин". Це невеликий пристрій, що підключається до паралельного комп'ютера. Існує кілька різновидів ключів з пам'яттю. Звертання до HASP відбувається винятково через API ключа і, до того ж, через ту саму крапку. Відповідно, туди ставиться підпрограма - емулятор ключа. Всі інші навороти не мають ніякого змісту. Крім того, можливе копіювання самого ключа чи апаратна эмуляція.

"Прив'язка" до комп'ютера.

"Прив'язка" програми до дистрибутивної дискети шляхом описаних раніше способів захисту несе в собі велика незручність для користувача, зв'язана з необхідністю роботи тільки з вставленої в дисковод оригінальною дискетою. Набагато зручніше мати необхідний програмний продукт записаним на вінчестері. Тому необхідно, щоб контролююча частина програми "запам'ятала" свій комп'ютер і потім при запуску порівнювала наявні характеристики з характеристиками "рідного" комп'ютера. У випадку

їхньої розбіжності можна вважати, що програма незаконно скопійована. Для цього треба знайти параметри, які б індивідуально характеризували кожну обчислювальну систему.

-
- Як видно з табл.1, тільки ті з програмістів спробують зняти захист із захищеного пакета, що мають власний інструментарій.
- Прикладами спеціальних оригінальних розробок є:
- 1) Емулятор середовища;
- 2) Симулятор мікропроцесора 8088 (фірми ЭЛИАС);
- 3) Спеціальний відладник для роботи в захищеному режимі на 386 процесорі.
-
- Крім названих основних характеристик систем захисту, кожна система може виконувати ряд сервісних функцій, що роблять її більш універсальною і зручною у роботі, але не підвищувальна надійність захисту.
- До таких функцій можна віднести наступні можливості:
- 1) інсталяція з декількох дискет;
- 2) реінсталляція;
- 3) завдання кількості інсталяцій;
- 4) інсталяція декількох програм з однієї дискети;
- 5) можливість інсталяції файлів розмір яких більше ємкості дискети;
- 6) прив'язка при інсталяції з дискети до апаратних особливостей ПЭВМ;
- 7) робота без ключової дискети;
- 8) установка числа запусків для захищених програм;
- 9) установка граничної дати;
- 10) використання пароля доступу до програм;
- 11) розміщення вірусу в захищеній програмі;
- 12) ідентифікація і лікування від вірусів;
- 13) візуалізація авторських прав;
- 14) неприпустимий формат дискет;
- 15) захист даних і графічних шрифтів;
- 16) можливість прив'язки до однієї ключової дискети декількох програм;
- 17) ведення бази даних по створеним ключової дискетам, захищеним програмам і покупцям.

Програмні засоби, що полегшують задачу

У принципі, кожну дискету з неушкодженою поверхнею можна скопіювати з використанням апаратури для аналогового копіювання дискет. Цими характеристиками володіє плата Copy II PC Option Board Deluxe. Що ж стосується налагодження, то маються апаратні відладники, і навіть програмні можуть використовувати додаткове hardware (наприклад, це може Periscope)

- Teledisk by Sydex - нескладний, але швидкий копировщик
-

Floppy Disk Analyser 6.0 пакет призначений для копіювання дискет, захищених від копіювання. Копіює всі існуючі в даний час системи захисту, включаючи CERBERUS 2.0 і Cop's COPYLOCK. Не працює в середовищі Windows.

Програма дозволяє не тільки точно копіювати захищені дискети, але також докладно аналізувати і досліджувати формат і дані всіх доріжок і секторів дискети, і при необхідності довільно модифікувати їх. Мається можливість самостійного конструювання нестандартних форматів і ключових міток будь-якої складності.

Лекція 6. Захист від НСК.

Програма може використовувати один чи кілька ознак для перевірки легальності копії. При цьому особливого значення набуває спосіб використання програми. Якщо програма розрахована на роботу на конкретному ПК, вибираються одні ознаки. Якщо вона може вільно переміщатися з одного комп'ютера на інший без утрати працездатності,- інші.

Програми першого типу назвемо стаціонарними, а другого - мобільними. Захист тих і інших програм має свої особливості, що ми будемо в міру необхідності обговорювати.

В усіх випадках перевірка легальності не повинна істотно позначатися на швидкодії програми чи жадати від користувача якихось додаткових дій. Наприклад, навряд чи можна вважати скільки-небудь ефективної систему, що використовує пароль. Система захисту не повинна перевіряти користувача, вона повинна перевірити копію. Тому звичайно ці перевірки використовуються в сполученні з перевірками інших індивідуальних ознак і призначені. Розглянемо популярні ознаки для захисту.

Захист, що спирається на твердий диск.

Захист по ключовій дискеті надає гарний рівень таємності, але він дуже стомлюючий і сильно набридає в щоденному використанні і може бути легко зруйнований при неправильному звертанні. Тому велика частина захищених пакетів може инсталюватися на твердий диск, використовуючи як захист мітку на твердому диску. Через існування різних типів фізичних інтерфейсів твердого диска, мітки на твердому диску звичайно є "найменшим загальним знаменником" різних можливостей і більш відкриті для втручання.

Обговоримо DC (контролер вінчестера) AT. Доступ до першого DC AT може здійснюватися по адресах 1F0-1F7 і 3F6. Другий DigitalController AT захоплює адреси 170-177 і 376. DC AT використовує IRQ 14 (int 76h у MS-DOS). Для низькорівневого доступу до HDD використовуються регістри:

1F0 - регістр даних, використовується для читання/запису вмісту сектора (512 байт). Хоча доступ до даних сектора може здійснюватися послівно, тільки байти є єдино прийнятною формою для запису ECC.

1F1 - читання: регістр прапорів помилки:

- AM даних не знайдена;
- погана рекалібровка;
- доріжка 0 не знайдена
- команда перервана;
- ID AM не знайдений;
- незрозуміла чи помилка;
- AM даних не знайдена;
- виявлено поганий блок.

1F1 - запис: початок припинення запису поточного циліндра / 4,0FFH забороняє використання цього.

1F2 - регістр підрахунку секторів. Використовується для завдання кількості секторів для передачі в багатосекторних операціях - 1, тобто значення 1 означає два сектори. DC може коректно обробляти більше секторів, чим міститься на одній доріжці, відповідно коректуючи номер голівки і циліндра. Під час операції форматування доріжки вказується кількість секторів, розташовуваних на одній доріжці (0FFH відповідає 255 секторам).

1F3 - регістр кількості секторів. Під час операції форматування доріжки задає значення GAP1 і GAP 3 мінус 3 байти.

1F4 - 8 молодших біт номера циліндра.

1F5 - 2 старших біти номера циліндра (використовуються біти 0-1). Деякі з контролерів допускають використання більш ніж двох бітов у цьому регістрі, підтримуючи в такий спосіб тверді диски більш ніж з 1023 циліндрами, але, у всякому разі, більшість BIOS використовує тільки ці два старших біти.

1F6 - вибір сектора/дисковода/голівки. В оригінальній специфікації 82062 було зарезервовано 3 біти для полючи вибору голівки і 2 біти для вибору дисковода, але ці значення були зігноровані зовнішніми схемами, так що реально це не має значення.

1F7 - читання: регістр стану. Біти цього регістра мають значення:

- сума помилок (OR усіх бітов у 1F1) ;
- команда виконується;
- дані ECC-коректні;
- запит даних (буфер очікує дані) ;
- пошук довершений;
- поганий запис;
- DC готовий;
- DC зайнятий (всі інші біти при цьому некоректні) .

1F7 - запис: регістр команди. Сюди записується код команди. Набір команд FDC AT (який перекриває набір 82062), використовуваний для роботи з твердим диском, включає наступні команди:

- Повернення на доріжку 0;
- Пошук;
- Читання сектора;
- Запис сектора;
- Переглядати ID;
- Форматування;
- Діагностика;
- Установка параметрів дисковода.

3F6 - запис: регістр режимів. Запис

- 02h знімає DC IRQ 14 із системної шини;
- 00h резюмує нормальні операції;
- 04h скидає контролер.

Характеристики сприятливі для нормальних операцій на комп'ютері, скорочують можливі перевірки для захисту від копіювання до додаткового/пропущеному сектору і порядку проходження секторів. Корекція даних ECC дозволяє доповнити способи захисту твердого диска третім варіантом, при якому біт даних змінюється в заданій позиції, і ховається за допомогою ECC.

Усі ці мітки можуть бути генеровані і перевірені при використанні для доступу до диска рівня BIOS, і, таким чином, доступ до DC на рівні чипів не має достатніх вигод, що виправдують відсутність сумісності.

Одержання інформації про характеристики дисковода

Для одержання інформації використовується команда "Identify drive". Ця команда застосовується в описуваній бібліотеці. Приклад її використання приведений нижче. Команда "Identify drive" дозволяє одержати різноманітну інформацію про пристрій (256 слів)

Дані, видавані контролером у відповідь на команду “Identify drive” мають наступну структуру – табл. 1.5.

Таблиця 1.5. Структура результату команди “Identify drive”

Слово	Призначення
0	Конфігурація
1	Число циліндрів
3	Число голівок (у мол. байте)
4	Сира ємність доріжки в байтах
5	Сира ємність сектора в байтах
6	Число секторів на доріжці (у мол. байте)
10-19	Серійний номер диска
22	Число байтів ECC у "довгих" командах
23-26	Номер ревізії контролера
27-46	Модель нагромаджувача
48	Дорівнює 1, якщо підтримується обмін подвійними словами;інакше 0
52	Те ж саме для DMA режимів
54	Число поточних циліндрів
55	Число поточних голівок
56	Число поточних секторів на доріжці
57-58	Поточна ємність нагромаджувача в секторах
60-61	Число адресуємих секторів у режимі LBA. Якщо LBA не підтримується, то тут звичайно міститься 0, але в деяких нагромаджувачах тут буває сміття. LBA-номер сектора, переданий у контролер, повинний бути в межах 0..n-1, де n – значення цього полючи

Серійний номер диска, модель і номер ревізії контролера (слова 10-19, 27-46 і 23-26 відповідно) - символьні (ASCII) рядка. Вони зберігаються з попарно переставленими буквами. Наприклад, якщо моделлю диска є ST31720A, те в полі моделі буде записано: TS1327A0. Ці поля можуть бути з правим чи лівим вирівнюванням із заповненням пробілами. Поточні параметри, що зберігаються в словах 54, 55, 56, 57-58, установлюються спеціальною командою контролера.

ATAPI-пристрою мають аналогічну команду "IDENTIFY PACKET DEVICE", що дозволяє одержати приблизно таку ж інформацію про пристрій з урахуванням особливостей ATAPI: тип пристрою (прямого доступу, послідовного доступу, принтер, процесор, CD-ROM, сканер, RAID і ін.)

Захист на рівні BIOS.

Очевидно, це базовий рівень для захисту твердого диска. Забезпечення скритності від простежування TSR програмами, що чіпляються до переривання 13H, може бути досягнутий в цьому випадку переглядом оброблювача int 13H доти, поки не буде знайдена точка входу в BIOS. Іншим способом обходу сторожів є можливо точний вимір інтервалів часу при визначеному чергуванні секторів.

Зміна чергування секторів.

Цей метод дуже схожий з описаним для дискет методом. На відміну від флоппі, де сектора, як правило, розташоване послідовно (чергування 1:1), сектора на твердому диску при низькорівневом форматуванні часто розташовуються так, щоб забезпечити можливо велику швидкість передачі даних. Унаслідок цього на твердому диску сутужніше помітити мітку чергування, чим на дискеті.

Зміна номерів секторів.

Цей метод повторює застосовуваний для дискет. Однак сектора на твердому диску звичайно щільно підігнані, так, що додаючи сектор з нестандартним номером, прийдеться видалити один із секторів даних. Така ситуація може бути легко виявлена.

Невикористовуванні області диска.

На рівні BIOS є дві не використовувані області майже на будь-якому твердому диску: на самому початку диска й у самому кінці. Перший сектор кожного твердого диска (крім SCSI чи ESDI) використовується під таблицю розділів, тоді як інші сектори нульового циліндра на доріжці 0 не використовуються за схемою розташування розділів фірм IBM і Microsoft. Однак ця область може бути використана програмами, що здійснюють власну розбивку на розділи.

Друга не використовувана область складається з користувальницького діагностичного циліндра, що є наступним після останнього циліндра диска на AT. Усе записане тут має дуже невеликі шанси на довге існування, оскільки будь-яка низкорівнева тестова програма (Norton DiskTreet) має право розпоряджатися цією доріжкою, як їй потрібно.

Залежність від номера кластера.

Стандартні засоби OS не дозволяють контролювати положення файлу кластер за кластером, так що ця інформація може бути використана для закодування образа програми і/чи даних. Номер стартового кластера може бути отриманий викликом функції 11H. Знайти номера інших кластерів можна, переглядаючи FAT.

Невикористовуванні (зарезервовані) області диска.

У дійсності мається тільки одна така область зі зсувом 0Ch у таблиці опису файлу (довжиною 10 байт). На жаль, ця область часто використовується "сумісними з OS" операційними системами. Приміром, використовують це поле для збереження файлового пароля, зберігають тут ID власника файлу, права доступу і дату і час створення файлу.

Іншою не використовуваною областю, що може існувати на диску, є залишок останнього сектора в FAT 1, що резервується OS (і дорівнює копіюється в усі інші копії FAT).

Невикористовуванні (внаслідок округлення кластера) області диска.

Оскільки OS розподіляє дисковий простір покластерно (кожен кластер містить 2^N секторів), а розмір файлу виміряється в байтах, більшість файлів має не використовуваний (і звичайно невидимий на рівні файлової системи) хвіст, що може бути використаний для цілей захисту. Дивна накладка в OS, що дозволяє переглянути (функція DOS 42h) дані за кінцем файлу, робить доступ до цього хвоста файлу досить простим для высокорівневих мов.

Захист на рівні OS.

Захист на рівні BIOS надає не так багато альтернатив, і OS поки має зручні (і відносно стерпні) способи доступу до розділів OS (використання int 25h/26h, що можуть бути використані для захисту від копіювання. Структура файлової системи OS загальновідома.

Захист, що спирається на системну плату і систему BIOS.

Іншою частиною PC, що разом із гнучкими і твердим диском, завжди доступна для захисту від копіювання, є системна плата. Хоча зараз материнські плати виготовляються серіями, майже кожна з них має (чи здобуває) індивідуальні якості.

Способи, що базуються на даних.

Кожна системна плата має власний BIOS. Це може використовуватися для захисту від копіювання, хоча ми можемо припустити, що в першу чергу це використовується виробниками апаратного забезпечення, а не незалежними виготовлювачами програм. Так, напханий партією зашифрованих і недокументованних функцій і таблиць системний BIOS, є різновид захисту від копіювання.

Іншою доступною областю даних на системній платі є енергонезалежна пам'ять CMOS, що, як правило, має порцію (8 байт) невикористовуваного простору.

Способи, що базуються на тимчасових параметрах.

Більш цікавий спосіб захисту спирається як на визначенні характеристик системної плати, так і на вимірі внутрішніх тимчасових параметрів. Три основні підсистеми доступні для таких вимірів:

- CPU,
- пам'ять,
- підсистема введення/висновку.

Фактичні розходження за цими показниками можуть бути досить великими.

Не секрет, що програмний пакет Microsoft Office є самим популярної і найбільш використовуваним для підготовки документів. При роботі з додатками MS Office виникає проблема забезпечення конфіденційності інформації, що зберігається в документах. На жаль не всі способи захисту, реалізовані в цьому пакеті, дозволяють надійно захистити інформацію. Розглянемо криптостійкість захисту від Microsoft на прикладі **захисту документів** Microsoft Word. Цей спосіб захисту є самим слабким. Пароль захисту запису зберігається в документі в чистому виді. Можна навіть пошукати його будь-яким hex-редактором. Зберігається він у unicode. Пароль навіть не спромоглись захешувати. Зняти цей "захист" можна зміною одного біта в документі.

пароль "захисту документа від змін". Його криптостійкість не набагато відрізняється від попереднього пароля. Відмінність тільки в тім, що цей пароль хешується. Довжина хешу - 32 біта. Можна або замінити хеш на заздалегідь відомий, або обчислити перший придатний пароль. Зрозуміло що для такої довжини хешу придатних паролів може бути кілька.

Пароль на документ. Із всіх існуючих способів захисту документів Word цей є самим стійкої. При установці цього пароля документ шифрується по симметричному алгоритму RC4. У документі зберігається зашифрований хеш пароля, використовуваний при перевірці. Єдиний спосіб злому пароля - перебір. Однак експортні обмеження на кріптоалгоритмы значно знижують криптостійкость цього захисту. Ключ, використовуваний при шифруванні RC4, має довжину 40 біт. Якщо перебирати не паролі, а ключі, шуканий ключ можна знайти за 30 днів на P-II 350. У документах Office 9x використовується більш простий спосіб шифрування, що дозволяє знайти пароль будь-якої довжини майже миттєво.

Ще один спосіб захисту від несанкціонованого використання інформації називається **опитуванням довідників**. Він не забезпечує стійкого, довгострокового захисту програмного забезпечення, але може бути легко реалізований і дуже зручний у роботі. При його застосуванні користувач не зобов'язаний працювати з дистрибутивною дискетою, можна і з її копіями. Робота програм і їхніх копій, захищених на основі цього методу, поділяється на два етапи.

Перший: одержання від користувача інформації з деякого джерела, називаного довідником, і порівняння її з еталонною. Тобто програма перевіряє, чи дійсно введена інформація знаходиться в довіднику.

Другий: робота самої програми чи її копії. При цьому необхідно зробити все можливе, щоб текст довідника був доступний тільки законному користувачу, тому що доступ до довідника в цьому методі знімає всі перешкоди для роботи програми.

У найпростішому випадку довідником є звичайний текстовий файл, що користувач розташовує на дискеті, на твердому диску, або має його як роздруківку.

Це дозволить користувачу не тримати в пам'яті інформацію про те, яке слово де знаходиться, і не витрачати час на пошуки потрібного слова в довіднику.

Однак перегляд перед кожним запуском програми списку паролів і випробування кожного з них займає досить багато часу. До того ж у таких системах часто виникають помилки через неоднозначність рішення: виділяти чи ні заголовок як окремий абзац. Наприклад, узявши п'яте слово з третього абзацу як пароль, ви не можете запустити програму, тому що автор виділив заголовок в окремий абзац і необхідне слово знаходиться не в третьому, а в другому абзаці тексту, якщо розглядати його без заголовка.

Через зазначені недоліки описаний вище механізм найчастіше використовуються як складену частину багатопрофільних систем захисту від копіювання.

Введення обмежень на використання програмного забезпечення

В одному зі способів захисту програмного забезпечення від незаконного використання застосовуються обмеження по:

часу його експлуатації;

кількості запусків;

його переміщенню для використання на інших машинах.

Ще одним способом обмежити використання програмного забезпечення є введення лічильника запусків. Звичайно такий лічильник встановлюють у:

CMOS пам'яті;

тілі чи програми в спеціальних числових файлах;

завантажувальному секторі диска(замість інформації про версію DOS);

FAT (у першому елементі таблиці, звичайно заповненому кодом FFF);

резервних секторах;

збійних секторах;

системному реєстрі;

за кінцем одного з файлів на локальному диску.

Зменшуючи значення лічильника при кожнім запуску, контрольна частина програми стежить за ним і при досягненні нульового значення виконує передбачені розроблювачем дії.

Визначення ключової інформації про систему в середовищі Windows **може бути виконане за допомогою наступних Win API функцій (інформація про систему)**

1) GetLogicalDrives. Функція GetLogicalDrives повертає чисельно-бітову маску в який зберігаються всі доступні диски.

DWORD GetLogicalDrives(VOID);

Ця функція не має параметрів. Якщо функція викликана правильно, то вона повертає чисельно-бітову маску в який зберігаються всі доступні диски (якщо 0 біт дорівнює 1, то диск "A:" є присутнім, і т.д.). Якщо функція викликана не правильно, то вона повертає 0. Приклад:

```
int n;
char dd[4];
DWORD dr = GetLogicalDrives();

for( int i = 0; i < 26; i++ )
{
 n = ((dr>>i)&0x00000001);
 if( n == 1 )
 {
   dd[0] =  char(65+i);
   dd[1] = ':';
   dd[2] = '\\';
   dd[3] = 0;
  cout << "Available disk drives : " << dd << endl;
  }
}
```

2) GetDriveType повертає тип диска (removable, fixed, CD-ROM, RAM disk, чи network drive).

UINT GetDriveType(LPCTSTR lpRootPathName);

Параметри:

LpRootPathName [in] Покажчик на ненульову стоку в який зберігається ім'я головної директорії на диску. Зворотний слэш потрібен. Якщо lpRootPathName дорівнює NULL, то функція використовує поточну директорію.

Функція повертає тип диска. Можуть бути наступні значення:

Значення	Опис
DRIVE_UNKNOWN	Не відомий тип.
DRIVE_NO_ROOT_DIR	Не правильний шлях.
DRIVE_REMOVABLE	Знімний диск.
DRIVE_FIXED	Фіксований диск.
DRIVE_REMOTE	Вилучений або network диск.
DRIVE_CDROM	CD-ROM диск.

DRIVE_RAMDISK	RAM диск.

Приклад:

```
int d;
d = GetDriveType( "c:\\" );
if( d == DRIVE_UNKNOWN ) cout << " UNKNOWN" << endl;
if( d == DRIVE_NO_ROOT_DIR ) cout << " DRIVE NO ROOT DIR" <<
endl;
if( d == DRIVE_REMOVABLE ) cout << " REMOVABLE" << endl;
if( d == DRIVE_FIXED ) cout << " FIXED" << endl;
if( d == DRIVE_REMOTE ) cout << " REMOTE" << endl;
if( d == DRIVE_CDROM ) cout << " CDROM" << endl;
if( d == DRIVE_RAMDISK ) cout << " RAMDISK" << endl;
```

3) GetVolumeInformation. Функція GetVolumeInformation повертає інформацію про файлову систему і диски(директоріях).

```
BOOL GetVolumeInformation(
  LPCTSTR lpRootPathName,      // ім'я диска(директорії)[in]
  LPTSTR lpVolumeNameBuffer,   // назва диска       [out]
  DWORD nVolumeNameSize,       // довжина буфера назви диска [in]
  LPDWORD lpVolumeSerialNumber,// серійний номер диска     [out]
  LPDWORD lpMaximumComponentLength,// максимальна довжина файлу     [out]
  LPDWORD lpFileSystemFlags,   // опції файлової системи     [out]
  LPTSTR lpFileSystemNameBuffer,// ім'я файлової системи      [out]
  DWORD nFileSystemNameSize     // довжина буфера імені файл. [in]
);
```

Якщо функція викликана правильно, те вона повертає не нульове значення(TRUE). Якщо функція викликана не правильно, то вона повертає 0(FALSE).Приклад:

```
   char VolumeNameBuffer[100];
   char FileSystemNameBuffer[100];
   unsigned long VolumeSerialNumber;

BOOL GetVolumeInformationFlag = GetVolumeInformation(
   "c:\\",
   VolumeNameBuffer,
   100,
   &VolumeSerialNumber,
   NULL, //&MaximumComponentLength,
   NULL, //&FileSystemFlags,
   FileSystemNameBuffer,
   100
   );

if(GetVolumeInformationFlag != 0)
{
 cout << "Volume Name is " << VolumeNameBuffer << endl;
 cout << "Volume Serial Number is " << VolumeSerialNumber <<
endl;
 cout << "File System is " << FileSystemNameBuffer << endl;
  }
```

```
else cout << "  Not Present (GetVolumeInformation)" << endl;
```

Для перевірки, чи є дискета в дисководі виконують наступний спосіб.

```
    BOOL IsDiskInDrive(LPTSTR lpszDrive)
 {
  UINT  errmode;
  TCHAR szVolName[256];
  DWORD dwMaxComSize;
  DWORD dwFlags;
  TCHAR szFS[256];
  BOOL  bRes;
  errmode = SetErrorMode(SEM_FAILCRITICALERRORS);
 // якщо не змінити режим повідомлень про помилки
 // з'явиться стандартне вікно "Drive Not Ready"
   bRes = GetVolumeInformation(lpszDrive, szVolName,
sizeof(szVolName), NULL,&dwMaxComSize, &dwFlags, szFS,
sizeof(szFS));
   SetErrorMode(errmode);
   return bRes;
 }
```

Ця технологія працює з CDROM і іншими змінними пристроями.

4) GetDiskFreeSpaceEx. Функція GetDiskFreeSpaceEx видає інформацію про доступне місце на диску.

```
BOOL GetDiskFreeSpaceEx(
LPCTSTR lpDirectoryName,   // ім'я диска(директорії)[in]
   PULARGE_INTEGER lpFreeBytesAvailable,/*доступно для
використання(байт) [out]*/
PULARGE_INTEGER lpTotalNumberOfBytes,// максимальний обсяг(байт)[out]
PULARGE_INTEGER lpTotalNumberOfFreeBytes//вільно на диску(байт)[out]
);
```

Якщо функція викликана правильно, те вона повертає не нульове значення (TRUE). Якщо функція викликана не правильно, то вона повертає 0(FALSE). Приклад:

```
DWORD FreeBytesAvailable;
DWORD TotalNumberOfBytes;
DWORD TotalNumberOfFreeBytes;

BOOL GetDiskFreeSpaceFlag = GetDiskFreeSpaceEx(
"c:\\",                                  // directory name
(PULARGE_INTEGER)&FreeBytesAvailable,     // bytes available to
caller
(PULARGE_INTEGER)&TotalNumberOfBytes,     // bytes on disk
(PULARGE_INTEGER)&TotalNumberOfFreeBytes  // free bytes on disk
);

if(GetDiskFreeSpaceFlag != 0)
{
      cout << "Total Number Of Free Bytes = " <<
        (unsigned long)TotalNumberOfFreeBytes       << "( " <<
   double(unsigned long(TotalNumberOfFreeBytes))/1024/1000 <<"
Mb )" << endl;
cout << "Total Number Of Bytes = " <<
```

```
    (unsigned long)TotalNumberOfBytes  <<
    "( " << double(unsigned long(TotalNumberOfBytes))/1024/1000 <<
    " Mb )" << endl;
}
else      cout << "     Not Present (GetDiskFreeSpace)" << endl;
```

5) GetComputerName, GetUserName. Функція GetComputerName повертає NetBIOS ім'я локального комп'ютера.

BOOL GetComputerName(
LPTSTR lpBuffer, // ім'я локального комп'ютера(довжина
буфера дорівнює MAX_COMPUTERNAME_LENGTH + 1) [out]
LPDWORD lpnSize // розмір буфера (краще поставити
MAX_COMPUTERNAME_LENGTH + 1) [out/in]
);

6). Функція GetUserName повертає ім'я поточного користувача.

BOOL GetUserName(
LPTSTR lpBuffer, // ім'я користувача (довжина буфера дорівнює UNLEN + 1) [out]
LPDWORD nSize // розмір буфера (краще поставити UNLEN + 1) [out/in]
);

Якщо функції викликані правильно, те вони повертають не нульове значення(TRUE).Якщо функції викликані не правильно, то вони повертають 0 (FALSE).
Приклад:

```
char ComputerName[MAX_COMPUTERNAME_LENGTH + 1];
unsigned long len_ComputerName = MAX_COMPUTERNAME_LENGTH + 1;
char UserName[UNLEN + 1];
unsigned long len_UserName = UNLEN + 1;

BOOL comp = GetComputerName(
    ComputerName,
    &len_ComputerName
    );
if(comp != 0) {cout <<"Computer Name is"<<ComputerName<< endl;}
else cout << "Computer Name is NOT FOUND !!! " << endl;
comp = GetUserNameA (
    UserName,
    &len_UserName
    );
if( comp != 0 ) { cout << "User Name is " << UserName << endl; }
else cout << "User Name is NOT FOUND !!! " << endl;
```

7) GetSystemDirectory, GetTempPath, GetWindowsDirectory, GetCurrentDirectory

Функція GetSystemDirectory повертає шлях до системної директорії.

```
UINT GetSystemDirectory(
 LPTSTR lpBuffer,  // буфер для системної директорії      [out]
 UINT uSize       // розмір буфера                        [in]
);
```

Ця функція повертає розмір буфера для системної директорії не включаючи нульового значення наприкінці, якщо вона викликана правильно.
Якщо функція викликана не правильно, то вона повертає 0.
Функція GetCurrentDirectory повертає шлях до поточного директорії.

```
DWORD GetCurrentDirectory(
 DWORD nBufferLength,  // розмір буфера                [in]
 LPTSTR lpBuffer       // буфер для поточної директорії [out]
);
```

Ця функція повертає розмір буфера для системної директорії не включаючи нульового значення наприкінці, якщо вона викликана правильно.
Якщо функція викликана не правильно, то вона повертає 0.
Приклад:

```
char path[100];
. . . . . .
GetSystemDirectory( path, 100 );
cout << "System Directory is " << path << endl;
GetTempPath( 100, path );
cout << "Temp path is " << path << endl;
GetWindowsDirectory( path, 100 );
cout << "Windows directory is " << path << endl;
GetCurrentDirectory( 100, path );
cout << "Current directory is " << path << endl;
```

Лекція 7. Збереження ключової інформації

Ключову інформацію (для стислості - ключ) про захищаєму програму можна зберігати:

- у невикористовуваній області простору FDD чи HDD, особливо використовуючи знання за структурою доріжки;
- у невикористовуваній області збереження файлу;
- в зовнішньому файлі;
- у самій програмі, що захищається.

Розглянемо приклад другого способу. Програма записує ключ за кінцем файлу.

```
#include <string.h>
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>

int main(void)
{
   int handle,n=0;
   long curpos=0,length=0;
   char msg[] = "Hello world";
```

/*open opens the file specified by path(TEST.$$$), then prepares it for reading and/or writing as determined by the value of access.
O_CREAT - If the file exists, this flag has no effect. If the file does not exist, the file is created, and the bits of mode are used to set the file attribute bits as in chmod.
O_BINARY This flag can be given to explicitly open the file in binary mode.If the O_CREAT flag is used in constructing access, you need to supply the mode argument to open from the following symbolic constants defined in sys\stat.h.

Value of mode	Access permission
S_IWRITE	Permission to write
S_IREAD	Permission to read
S_IREAD|S_IWRITE	Permission to read/write

Return Value:
On successful completion, open returns a nonnegative integer (the file handle).*/

```
   if ((handle = open("TEST.$$$", O_CREAT | O_WRONLY | O_BINARY,
                   S_IWRITE | S_IREAD)) == -1)
    {
      perror("Error:");
      return 1;
     }
```

/*lseek sets the file pointer associated with handle to a new position offset
bytes beyond the file location given by 3-d parameter.

3-parameter	File location
SEEK_SET (0)	File beginning
SEEK_CUR (1)	Current file pointer position
SEEK_END (2)	End-of-file */

```
   curpos=lseek(handle,0L,SEEK_SET);
   printf (" begin of files=%ld \n",curpos);
```

```
    lseek(handle,0L,SEEK_END);
```
/*tell gets the current position of the file pointer associated with handle and expresses it as the number of bytes from the beginning of the file.*/

```
    length = tell(handle);
    printf (" Length of files =%ld \n", length);
```

// Calculates length of a string

```
    n = strlen(msg);
```
/*next function attempts to write n bytes from the buffer pointed to msg to the file associated with handle*/

```
    write(handle, msg,n);
    printf("Write message  !!%s!!  after end of file", msg);

    printf ("truncate file after writing \n");
```
//changes the size of the file associated with handle

```
    chsize(handle,length);
    close(handle);
    return 0;
}
```

Ключову інформацію про незаражену програму можна зберігати в окремому файлі. При розробці алгоритмів ставлять задачу помістити у файл WAV– формату ключову інформацію таким чином, щоб файл зберіг можливість програвання, але за допомогою розробленої програми інформацію можна було витягти з файлу. Для побудови алгоритму необхідно визначити формат WAV – файлу (табл.2.1.).

Як видно з таблиці, довжина кожної частини файлу визначається безпосередньо перед нею. Проведені дослідження, показали, що внесення даних між блоками з маніпуляцією значеннями їхніх довжин не дозволяє зберегти цілісність файлу, і він стає недоступним для програвання. У той же час виявлено, що крім описаних у таблиці елементів файл wav-формату може мати і додаткові полючи.

Таким полем є блок, що йде наприкінці файлу і починається з ідентифікатора LIST. Далі у форматі 4б випливає довжина додаткового блоку, і полючи з інформацією. Максимальне число полів не встановлено, але поле завжди повинне починатися з зарезервованого слова:

ICRD – дата створення
IART – виконавець
IKEY – ключ
ISRF – програма творець
ICOM – коментарі
…

За кожним словом випливає довжина полю у форматі 4б і саме поле, що завершується 0. У принципі можливий запис інформації , наприклад, у виді коментарю, але недолік такого методу в тім, що інформація може бути прочитана за допомогою багатьох програм обробки звукових файлів, наприклад, COOLEDIT і т.п. У такий спосіб доцільним представляється записувати інформацію в область, не сприйману іншими програмами як область даних.

Найбільше просто це реалізується простим дописуванням інформації (закодованої) у кінець файлу. У цьому випадку звичайні програми, що визначають довжину блоку даних по заголовку, не сприймуть нова ділянка як дані, у той же час наша програма може визначити

наявність даного блоку і прочитати його. Для більшої вірогідності перед блоку даних можна помістити сигнатуру (байт FF).

Таблиця 2.1

Зсув	Довжина	Опис
0h	4	Ідентифікатор формату RIFF
4h	4	Довжина блоку даних
8h	4	Ідентифікатор блоку звукових даних WAVE
0Ch	4	Ідентифікатор підблоку заголовка fmt
10h	4	довжина підблоку заголовка
14h	2	тип формату представлення даних
16h	2	число каналів
18h	2	частота дискретизації
1Ah	2	швидкість передачі даних
1Ch	2	число байт для представлення одного відліку
1Eh	2	Розрядність
20h	4	ідентифікатор підблоку даних data
24h	4	довжина звукових даних
28h		звукові дані

Або у запису на мові С:

```
typedef struct {
    char id[4]; - ID of this file = "RIFF" = 0x46464952
    long len;   - Length of this file without of this header
} IDRiff;
typedef struct {
  char id[4];  - ID = "WAVE" = 0x45564157
  char fmt[4]; - ID = "fmt " = 0x20746D66
  long len;    - Length of next part of WAV chunk,usually-16
} IDChuckWave;
typedef struct {
```

int type; - type of voice date (Pulse code Modulation,
regular 8 bit sampled uncompressed sound):
1 – sample of Microsoft;
0x101 - IBM mu-law;
0x102 - IBM a-law;
0x103 - ADPCM.

int channels; - number of channels 1=mono/2=stereo. In mono 8-bit files each byte *represents* one sample. In stereo 8-bit files two bytes *are stored* for each sample, the first byte *is* the left channel value, the next *is* the right channel value.

long SamplesPerSec; - Playback frequency(частота вибірки)
long AvgBytesPerSec; -sample per second or average number of bytes per second *should be transferred*(частота видачі байтів)
int align; -block alignment (вирівнювання)
int bits; -Bit per sample
} IDWave;
Data Chunk *is placed* after that which *is split up* into these fields:
typedef struct {
char id[4]; -*contains* the characters of data"=0x61746164
long len; - Length of data into sample (кратно 2)

} IDSampleWave;

Then *is placed* the actual waveform data.

Програма для читання WAV-file:

```
#include <stdio.h>
#include <string.h>
typedef struct {
     char id_riff[4];
     long len_riff;
     char id_chuck[4];
     char fmt[4];
     long len_chuck;
     int  type;
     int  channels;
     long freq;
     long bytes;
     int  align;
     int  bits;
     char id_data[4];
     long len_data;
} TitleWave;
void    main
     ( int argc, char * argv[] )
{
FILE * f;
TitleWave tw;
clrscr;
if ( argc<2 ) { printf("Вкажи ім'я .wav файлу\n"); return ; }
f = fopen(argv[1],"rb");
if  ( f  ==  0  )  {  printf("Не  відкрити  файл  -  %s\n",  argv[1]);
return; }
fread(&tw,sizeof(TitleWave),1,f);
fclose(f);
printf("LEN RIFF\t - %ld\n", tw.len_riff );
if ( strncmp(tw.id_riff,"RIFF",4)!=0 )
     printf("Не збігся ідентифікатор RIFF\n");
printf("LEN Chuck\t - %ld\n", tw.len_chunk );
if ( strncmp(tw.id_chuck,"WAVE",4)!=0 )
     printf("Не збігся ідентифікатор CHUnk\n");
if ( strncmp(tw.fmt,"fmt ",4)!=0 )
     printf("Не збігся ідентифікатор FMT\n");
printf("Type\t\t - %d\n", tw.type );
printf("Channels\t - %d\n", tw.channels );
printf("Sample Per Sec\t - %d\n", tw.freq );
printf("Bytes Per Sec\t - %d\n", tw.bytes );
printf("Bits\t\t - %d\n", tw.bits );
printf("Aligned\t\t - %d\n", tw.align );
printf("LEN Data\t - %ld\n", tw.len_data );
if ( strncmp(tw.id_data,"data",4)!=0 )
     printf("Не збігся ідентифікатор DATA\n");
}
```

Для запису ключа у такий файл треба виконати наступне.

```
tw.len_data  = any value;
```

```
/*PadFl is used for aligment*/
        if( tw.len_data  % 2 ) PadFlg=1;
        else PadFlg=0;

        tw.len_riff=sizeof(Wave)+sizeof(Data)+DataLen+PadFlg;
        Wave.SamplesPerSecond = any value;
        Wave.AvgBytesPerSecond=any value;
        Data.Len=DataLen+PadFlg;

        fwrite(&Riff,1,sizeof(Riff),fo);
        fwrite(&Wave,1,sizeof(Wave),fo);
        fwrite(&Data,1,sizeof(Data),fo);

        {
                unsigned c;
                for(long i=0; i<DataLen; i++)
                {
                        c=getc(fi);
                        putc(c,fo);
                }
                if( PadFlg ) putc(c,fo);
        }
```

Іншим прикладом може бути графічний файл. Формат графічного файлу GIF {GENERAL FILE FORMAT):

- GIF Signature
- Screen Descriptor
- Global Color Map
- Image Descriptor
- Local Color Map | |- Repeated 1 to n times
- Raster Data
- ... ...
- GIF Terminator

При збереженні ключа існує небезпека втратити додатковий файл при копіюванні програми чи помилково знищити його. Набагато надійніше зберегти ключ у тілі самого файлу, який захищається, На жаль, його не можна подібно вірусу пристикувати в кінець файлу, У випадку зараження вірус змінить поля *PartPag* і *PageCnt* і ми не зможемо визначити те місце у файлі, де він розташовується.

Треба врахувати, що всі константи (у тому числі і типізовані) створюються на етапі компіляції програми, таким чином, у файлі обов'язково мається область даних, що містить значення цих констант. Ця область у програмах (Турбо Паскаль) розташовується в самому кінці частини файлу, який завантажується.

Отже, ми повинні визначити в програмі типізовану константу, призначену для збереження ключа. Потім в область файлу, відведену для її розміщення, помістити потрібну інформацію.

Яким образом відшукати в *EXE* - файлі місце, займане ключем? Звичайно, можна перед ним у програмі розмістити яку-небудь типізовану константу з характерним значенням (наприклад, заздалегідь обумовлений текстовий рядок) і потім відшукувати її у файлі, Однак таке рішення навряд чи задовільне:

- по-перше, завжди існує імовірність того, що якийсь фрагмент кодів програми містить той же ланцюжок байт, що і заголовок ключа;
- по-друге, прийдеться переглядати часом великий по обсязі *EXE*—файл у пошуках потрібної константи.

Значно надійніше виглядає рішення, засноване на точному обчисленні зсуву від початку файлу до ключа. Для цього ми можемо використовувати операцію одержання адреси @. Результатом застосування цієї операції до адреси константи, є покажчик (4 -бітна адреса). Зсув адреси, що він містить, і є потрібним нам зсувом початку ключа відносно початку області даних..

```
uses crt;
{-------------------------------------}
procedure WriteHexWord(w: Word);  {процедура обчислення HEX слова}
const
  hexChars: array [0..$F] of Char =  '0123456789ABCDEF';
begin
  Write(hexChars[Hi(w) shr 4],
        hexChars[Hi(w) and $F],
        hexChars[Lo(w) shr 4],
        hexChars[Lo(w) and $F]);
end;
{головна програма }
var
  i: Integer;   {для неї друкуємо адреса}
begin
  clrscr;
{---------------------------------------}
{друкуємо сегмент коду}
  Write('The current code segment is $');
  WriteHexWord(CSeg); WriteLn;
{друкуємо сегмент даних}
  Write('The global data segment is $');
  WriteHexWord(DSeg); WriteLn;
{друкуємо сегмент стека}
  Write('The stack segment is $');
  WriteHexWord(SSeg); WriteLn;
{друкуємо покажчик на стек}
  Write('The stack pointer is at $');
  WriteHexWord(SPtr); WriteLn;
{//////////////////////////////////////////}
{друкуємо зсув для i}
  Write('i is at offset $');
  WriteHexWord(Ofs(i));
```

```
{друкуємо сегиент для i}
  Write(' in segment $');
  WriteHexWord(Seg(i));
end.
```

У заключенні розглянемо приклад програми роботи з диском використовуючи сервіс DeviceIoControl

```
/*******************************************************
#include <windows.h>
#define VWIN32_DIOC_DOS_INT25 2
#define VWIN32_DIOC_DOS_INT26 3
#define VWIN32_DIOC_DOS_DRIVEINFO 6
typedef struct _DIOC_REGISTERS {
     DWORD reg_EBX;
     DWORD reg_EDX;
     DWORD reg_ECX;
     DWORD reg_EAX;
     DWORD reg_EDI;
     DWORD reg_ESI;
     DWORD reg_Flags;
} DIOC_REGISTERS, *PDIOC_REGISTERS;
 #define CARRY_FLAG 1
 #pragma pack(1)
typedef struct _DISKIO {
 DWORD dwStartSector; // starting logical sector number
 WORD wSectors; // number of sectors
 DWORD dwBuffer; // address of read/write buffer
 } DISKIO, * PDISKIO;
#pragma pack()
/*---------опис функції----------------------------------------------------
ReadLogicalSectors (hDev, bDrive, dwStartSector, wSectors,
        lpSectBuff)
Purpose: Reads sectors from a logical drive. Uses Int 25h.
 Parameters: hDev Handle of VWIN32
bDrive- The MS-DOS logical drive number. 1 = A, 2 = B, 3 = C, etc.
dwStartSector- The first logical sector to read
wSectors     = The number of sectors to read
lpSectBuff   - The caller-supplied buffer that will contain the sector data
Return Value: Returns TRUE if successful, or FALSE if failure.
Comments: This function does not validate its parameters.
-------------------------------------------------------------*/
 BOOL ReadLogicalSectors (HANDLE hDev,
 BYTE bDrive,
 DWORD dwStartSector,
 WORD wSectors,
 LPBYTE lpSectBuff)
{
BOOL fResult;
DWORD cb;
DIOC_REGISTERS reg = {0};
DISKIO dio = {0};
dio.dwStartSector = dwStartSector;
dio.wSectors = wSectors;
```

```
dio.dwBuffer = (DWORD)lpSectBuff;
reg.reg_EAX = bDrive - 1; // Int 25h drive numbers are 0-based.
reg.reg_EBX = (DWORD)&dio;
reg.reg_ECX = 0xFFFF; // use DISKIO struct
fResult = DeviceIoControl (hDev, VWIN32_DIOC_DOS_INT25,
            &reg, sizeof(reg),
            &reg, sizeof(reg), &cb, 0);
// Determine if the DeviceIoControl call and the read succeeded.
fResult = fResult && !(reg.reg_Flags & CARRY_FLAG); return
fResult;
}
/*---- опис функції ----------------------------------------
WriteLogicalSectors (hDev, bDrive, dwStartSector, wSectors,
      lpSectBuff)
Purpose: Writes sectors to a logical drive.
Uses Int 26h Parameters:
    hDev Handle of VWIN32
    bDrive- The MS-DOS logical drive number. 1 = A, 2 = B, 3 = C, etc.
    dwStartSector- The first logical sector to write
    wSectors- The number of sectors to write
    lpSectBuff- The caller-supplied buffer that contains the sector data
Return Value: Returns TRUE if successful, or FALSE if failure.
Comments: This function does not validate its parameters.
-----------------------------------------------------------*/
BOOL WriteLogicalSectors (HANDLE hDev,
            BYTE bDrive,
            DWORD dwStartSector,
            WORD wSectors,
            LPBYTE lpSectBuff)
{
 BOOL fResult;
 DWORD cb;
 DIOC_REGISTERS reg = {0};
 DISKIO dio = {0};
 dio.dwStartSector = dwStartSector;
 dio.wSectors = wSectors;
 dio.dwBuffer = (DWORD)lpSectBuff;
 reg.reg_EAX = bDrive - 1; // Int 26h drive numbers are 0-based.
 reg.reg_EBX = (DWORD)&dio;
 reg.reg_ECX = 0xFFFF; // use DISKIO struct
 fResult = DeviceIoControl(hDev, VWIN32_DIOC_DOS_INT26,
            &reg, sizeof(reg),
            &reg, sizeof(reg), &cb, 0);
// Determine if the DeviceIoControl call and the write succeeded.
fResult = fResult && !(reg.reg_Flags & CARRY_FLAG); return
fResult;
}
/*---------- опис функції -----------------------------------------
  NewReadSectors(hDev, bDrive, dwStartSector, wSectors, lpSectBuff)
    Purpose: Reads the specified number of sectors into a caller-supplied buffer. Uses Int 21h
function 7305h
Parameters: hDev- Handle of VWIN32
bDrive The MS-DOS logical drive number. 0 = default, 1 = A, 2 = B,
```

```
        3 = C, etc.
dwStartSector- The first sector to read.
WSectors     - The number of sectors to read.
LpSectBuff   - The caller-supplied buffer to read into.
Return Value: Returns TRUE if successful, or FALSE if failure.
Comments: This function does not validate its parameters. It assumes that lpSectBuff is allocated
by the caller and is large enough to hold all of the data from all of the sectors being read.
-----------------------------------------------------*/
BOOL NewReadSectors (HANDLE hDev, BYTE bDrive,
            DWORD dwStartSector,
            WORD wSectors,
            LPBYTE lpSectBuff)
{
BOOL fResult;
DWORD cb;
DIOC_REGISTERS reg = {0};
DISKIO dio;
dio.dwStartSector = dwStartSector;
dio.wSectors = wSectors;
dio.lpBuffer = (DWORD)lpSectBuff;
reg.reg_EAX = 0x7305; // Ext_ABSDiskReadWrite
reg.reg_EBX = (DWORD)&dio;
reg.reg_ECX = -1;
reg.reg_EDX = bDrive; // Int 21h, fn 7305h drive numbers are 1-
based
fResult = DeviceIoControl(hDev, VWIN32_DIOC_DOS_DRIVEINFO,
            &reg, sizeof(reg),
            &reg, sizeof(reg), &cb, 0);
// Determine if the DeviceIoControl call and the read succeeded.
fResult = fResult && !(reg.reg_Flags & CARRY_FLAG); return
fResult;
}
/*---------------- опис функції -------------------------------------
NewWriteSectors(hDev, bDrive, dwStartSector, wSectors, lpSectBuff)
Purpose:Writes the specified number of sectors from a caller-supplied buffer. Uses Int 21h function
7305h Parameters:
hDev- Handle of VWIN32
bDrive- The MS-DOS logical drive number. 0 = default, 1 = A, 2 = B, 3 = C, etc.
dwStartSector - The first sector to write.
wSectors The- number of sectors to write.
lpSectBuff- The caller-supplied buffer from which to write.
Return Value:  Returns TRUE if successful, or FALSE if failure.
Comments:This function does not validate its parameters. It assumes that lpSectBuff is allocated by
the caller and is large enough to hold all of the data to be written.
-----------------------------------------------------*/
     BOOL NewWriteSectors (HANDLE hDev, BYTE bDrive,
            DWORD dwStartSector,
            WORD wSectors,
            LPBYTE lpSectBuff)
{
BOOL fResult;
DWORD cb;
DIOC_REGISTERS reg = {0};
```

```
DISKIO dio;
dio.dwStartSector = dwStartSector;
dio.wSectors = wSectors;
dio.lpBuffer = (DWORD)lpSectBuff;
reg.reg_EAX = 0x7305; // Ext_ABSDiskReadWrite
reg.reg_EBX = (DWORD)&dio;
reg.reg_ECX = -1;
reg.reg_EDX = bDrive; // Int 21h, fn 7305h drive numbers are 1-
based
reg.reg_ESI = 0x6001; // Normal file data (See function
   // documentation for other values)
fResult = DeviceIoControl(hDev, VWIN32_DIOC_DOS_DRIVEINFO,
            &reg, sizeof(reg),
            &reg, sizeof(reg), &cb, 0);
 // Determine if the DeviceIoControl call and the write succeeded.
            fResult = fResult && !(reg.reg_Flags & CARRY_FLAG);
            return fResult; }
```

Лекція 8. Використання реєстру Windows для захисту даних.

Спочатку середовище Windows зверталася тільки до двох системних конфігураційних файлів - CONFIG.SYS і AUTOEXEC.BAT. У Windows 3.x до цих файлів додалися WIN.INI і SYSTEM.INI, а потім число конфігураційних файлів стало рости в міру появи реєстраційної бази OLE, файлу PROTOCOL.INI у системі Windows for Workgroups і безлічі INI-файлів для прикладних пакетів. Система Windows 9x може зчитувати ці конфігураційні файли, щоб забезпечувати сумісність зверху вниз, але їхні обов'язки тепер покладені на двійковий об'єкт, називаний реєстром (Registry).

Хоча реєстр усе ще поділений на кілька файлів, редактор реєстру Windows 95 (Registry Editor) звертається до нього як до єдиного цілого. Замість збереження інформації про настановні параметри у формі послідовного списку, у Windows 95 використовується ієрархічна деревоподібна структура, що заміняє прості записи в INI-файлі складними іменами типу MyComputer\HKEY_CLASSES_ROOT\CLSID\ {20D04FE0-3AEA-1069- A2D8-08002B30309D}\DefaultIcon.

Ієрархічна, централізована *база даних для Microsoft Windows 9x, Windows NT, Windows 2000 й Windows XP.*

У реєстрі зберігаються наступні дані, які необхідні для Windows:

- *профілі всіх користувачів,*
- *дані про встановлене програмне забезпечення й типи документів,,*
- *інформація про властивості папок,*
- *дані про встановлене встаткування й порти.*

Версії реєстру для різних версій операційних систем сімейства Windows мають ***розходження****.*

Любою процес, в ОС Windows, використає системний реєстр у своїх цілях. Реєстр зберігає *дані* , необхідні для програм, служб, драйверів. **Високорівневі** служби використають реєстр із самого початку завантаження Windows і до завершення роботи. Системний реєстр використається **драйверами** пристроїв для завантаження й ініціалізації.

Основу реєстру Windows - **кореневі** розділи. Кожен кореневий розділ має **вкладені** підрозділи й параметри – елементарні дані реєстру.

Він містить шістьох кореневих розділів, що формують перший рівень дерева. Всі інші розділи являють собою підрозділи цих шести:

- HKEY_LOCAL_MACHINE (HKLM) містить усі дані з System.dat, що описують драйвери, встановлене устаткування, комунікаційні порти, конфігурацію програмного забезпечення й інші елементи, що визначають індивідуальні параметри системи;
- HKEY_USERS містить підрозділи для використовуваного за замовчуванням блоку User.dat і підрозділи для файлу User.dat поточного користувача (якщо ПК набудований тільки для одного користувача, він буде містити тільки розділ "За замовчуванням"). У Windows Resource Kit затверджується, що HKEY_USERS містить підрозділи для усіх файлів User.dat, що розміщені в системі, але це не так;
- HKEY_CLASSES_ROOT (HKCR), що представляє собою копію роздягнула HKLM\SOFTWARE\Classes, містить інформацію про технологію OLE, системних абревіатурах, механізмі drag-and-drop і інших функціях інтерфейсу, реалізованих на рівні ядра системи;
- HKEY_CURRENT_USER (HKCU) містить дані з файлу User.dat поточного користувача. Отже, він завжди буде копією даних одного з підрозділів HKEY_USERS;

- HKEY_CURRENT_CONFIG містить інформацію про встановлені в даний момент апаратурних засобах. Оскільки HKLM\CONFIG охоплює всі можливі параметри устаткування системи, роздягнув HKEY_CURRENT_CONFIG завжди являє собою копію підрозділу з HKLM\CONFIG;
- HKEY_DYN_DATA, що зберігається в ОЗУ для забезпечення швидкого доступу до нього, містить інформацію про пристрої, що були встановлені чи завантажені - успішно або безуспішно.

Таким чином, у реєстрі є тільки три розділи, що не дублюються: HKEY_DYN_DATA, HKEY_USERS і HKEY_LOCAL_MACHINE. Отже, якщо ви експортуєте реєстр цілком, що виходить у результаті REG-файл містить лише два кореневих роздягнула - HKEY_USERS і HKEY_LOCAL_MACHINE (при цьому HKEY_DYN_DATA розміщається в ОЗУ).

Структура кореневих розділів може показатися надлишкової, але таке дублювання дуже корисне. Розділи HKEY_CURRENT_USER і HKEY_CURRENT_CONFIG дозволяють відокремити поточні настановні параметри вашої машини від інших, не завжди використовуваних характеристик системи. Роздягнув HKEY_CLASSES_ROOT - по суті копія всієї реєстраційної бази даних, що найбільш важлива при зміні настроювання системи. Саме тут визначається порядок роботи з файлами різних типів Windows 95.

Усі імена кореневих розділів починаються з рядка **HKEY_.** Це вказує на те, що це - дескриптор, якому можна використати програмою. Дескриптор - це значення, використовуване для унікального опису ресурсу, до якого програма може одержати доступ.

Реєстр Windows NT подібний до реєстру Windows 9x. Але є кілька **відмінностей**, основним з яких є організація **кореневих** розділів.

У реєстрі є шість розділів, але реально, розділ HKEY_DYN_DATA недоступний. Всі інші розділи не відрізняються від аналогічних в Windows 9x.

Друга відмінність - спосіб зберігання інформації реєстру. Якщо в Windows 9x вона зберігається у двох файлах: SYSTEM.DAT й USER.DAT, те системний реєстр NT Windows підрозділяється на складові частини, які називаються кущами або вуликами (hives). Кущ є файл, що містить кореневий розділ, його підрозділи й параметри. Відмінність кущів від інших розділів полягає в тому, що вони є постійними елементами реєстру. Кущі не створюються динамічно при завантаженні ОС і не віддаляються при її зупинці. В Windows 2000 файли всіх кущів реєстру, крім HKEY_CURRENT_USER, перебувають у папці **Windows\System32\Config.**

Допоміжні файли куща HKEY_CURRENT_USER зберігаються в папці **Windows\Profiles\Username.**

Розширення імен цих файлів указують на тип даних, що втримуються в них.

Кущу можуть відповідати й додаткові файли. У наступній таблиці перераховані стандартні кущі реєстру Windows 2000 і підтримуючі їхні файли.

HKEY_LOCAL_MACHINE \SAM	Містить інформацію SAM (Security Access Manager), що зберігається у файлах SAM, SAM.LOG, SAM.SAV у папці \ %Windows%\System32\Config.
HKEY_LOCAL_MACHINE \SECURITY	Містить інформацію безпеки у файлах SECURITY, SECURITY.LOG, SECURITY.SAV у папці \%Windows %\System32\Config.
HKEY_LOCAL_MACHINE \SOFTWARE	Містить інформацію про програмне забезпечення. Ця інформація зберігається у файлах SOFTWARE, SOFTWARE.LOG, SOFTWARE.SAV у папці \%Windows %\System32\Config.
HKEY_LOCAL_MACHINE	Містить інформацію про апаратні профілі цього підрозділу.

\SYSTEM	Інформація зберігається у файлах SYSTEM, SYSTEM.LOG, SYSTEM.SAV у папці \%Windows%\System32\Config.
HKEY_CURRENT_CONFIG	Містить інформацію про підрозділ System цього вулика, що зберігається у файлах SYSTEM.SAV й SYSTEM.ALT у папці \ %Windows%\System32\Config.
HKEY_USERS\.DEFAULT	Містить інформацію, що буде використатися для створення профілю нового користувача, що вперше реєструється в системі. Інформація зберігається у файлах DEFAULT, DEFAULT.LOG, DEFAULT.SAV у папці \%Windows %\System32\Config.
HKEY_CURRENT_USER	Містить інформацію про користувача, зареєстрованому в системі на сучасний момент. Ця інформація зберігається у файлах **NTUSER.DAT** й **NTUSER.**LOG, розташованих у каталозі \%Windows%\Profiles\Username, де Username - ім'я користувача, зареєстрованого в системі на даний момент.

У файлі Ntuser.dat зберігаються користувальницькі профілі. Файл Ntuser.dat.log відслідковує зміни, які вносилися у файл Ntuser.dat. Файли Ntuser й Userdiff були вперше уведені в Windows NT 4.0 і виконують наступні функції:

- Ntuser.dat - у файлі зберігається профіль користувача;
- файли Userdiff служать для відновлення профілів користувача, які використалися в більше ранніх версіях Windows NT.

Імена файлів вуликів розташовані в розділі HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\hivelist

Кущам відповідають файли чотирьох типів.

Тип файлу	Опис
Без розширення	Містить копію куща
Alt	Містить копію куща HKEY_LOCAL_MACHINE\System. Тільки розділ System має відповідний файл із розширенням alt
Log	Містить журнал транзакцій, у якому реєструються всі зміни, внесені в розділи.
Sav	Містить копії файлів куща в тім виді, що вони мали на момент завершення текстової фази процесу установки.

Відмінності реєстру Windows NT й 9x.

Дані реєстру зберігаються у вигляді параметрів, розташованих у відповідних розділах реєстру. Кожен параметр має: ім'я, тип даних і значення. Існують наступні основні типи даних у реєстрі Windows 2000:

Тип даних	Опис
REG_BINARY	Двійкові дані. Дані про апаратні компоненти зберігається у вигляді двійкових даних і виводиться в редакторі реєстру в шістнадцятирічному форматі.
REG_DWORD	Дані представлені значеннями, довжина яких

	дорівнює 4 байти. Цей тип даних використається для зберігання параметрів драйверів пристроїв і служб. Значення відображається редактором реєстру у двійковому, шістнадцятерічному або десятковому форматі.
REG_EXPAND_SZ 2000	Рядок даних змінної довжини. Цей тип даних мають змінні, при використанні програмою або службою.
REG_MULTI_SZ 2000	Багатостроковий текст. Цей тип мають списки. Записи розділяються пробілами, комами або іншими символами.
REG_SZ	Текстовий рядок фіксованої довжини.

Структура reg-файлів

Reg-файл - це файл, що має певну структуру й містить дані, які можуть бути імпортовані до реєстру. У першому рядку файлу обов'язково повинне бути уведене

REGEDIT4

Букви повинні бути більші. Іншого в першому рядку бути не повинне. Після цього тексту ОБОВ'ЯЗКОВО повинна бути порожній рядок. Потім, записується розділ реєстру, у якому треба вставити або змінити параметри. Назва роздягнула повинне бути записане у квадратних дужках [...]. Далі записуються параметри: один параметр у рядку. Якщо треба виконати зміни в декількох розділах, то треба залишити один порожній рядок перед наступним розділом. Наприклад:

```
REGEDIT4

[Razdel1]
"param1"="znachenie1"
"param2"="znachenei2"
"param3"="znachenie3"

[Razdel2]
"param_1"="znachenie_1"
```

Останній рядок у файлі повинна бути ПОРОЖНІЙ. Такий файл запускається як звичайна програма. Буде виданий запит про зміну в реєстрі. Після позитивної відповіді інформація з файлу буде імпортована. Про результати імпортування Windows повідомить у вікні.

У прикладі додаються параметри за допомогою рядків типу "param1"="znachenie1". У такий спосіб додається СТРОКОВИЙ параметр із ім'ям "param1" і значенням "znachenie1".

Для параметрів типу DWORD використається рядок

"param"=dword:XXXXXXXX

Тут "param" - ім'я параметра, dword - тип цього параметра (букви повинні бути маленькі!). Після двокрапки записане значення з восьми цифр у шістнадцятеричному (!) форматі. Більшість параметрів DWORD мають значення або 0, або 1, виходить, треба записати або 00000000, або 00000001 замість XXXXXXXX. Пробіли в рядку не дозволяються.
Для двійкового параметра формат запису:

"param"=hex:XX,XX,XX,....

Після знака "=" іде hex, указуючи, що це двійковий параметр. У реєстрі існують параметри "За замовчуванням" ("Default"). Для них треба додати такий рядок:

@="znachenie"

Приклад reg-файлу, що записує до реєстру сайт, домашньої сторінки в Internet Explorer'е:

```
REGEDIT4

[HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main]
    "Start Page" = http://www.win.ru/
```

Для видалення роздягнула з реєстру треба перед його ім'ям у квадратних дужках поставити символ "-". От як це виглядає:

```
[-HKEY_LOCAL_MACHINE\Software\QuickSoft\QuickStart]
```

За допомогою REG-файлів можна видаляти параметри. Синтаксис:

```
REGEDIT4

[HKEY_CURRENT_USER\Software]
"xxx"=-
```

Програмування для системного реєстру на C++

Розділ системного реєстру, є стандартним об'єктом виконавчої системи, що експортує підсистема Win32, і до якого можна одержати доступ через handle..

Для будь-якого об'єкта ядра потрібно одержати його опис і вказати дії, які будуть виконувати з ним. Опис розділу системного реєстру можна одержати створюючи або відкриваючи розділ. Для цього служать наступні Win32-функції.

RegOpenKey
RegOpenKeyEx
RegCreateKeyEx

Як видно з назви функції *RegOpenKey* й *RegOpenKeyEx* відкривають розділ реєстру (одержують опис), а *RegCreateKeyEx* - створює розділ реєстру й теж одержує опис.

***LONG RegOpenKey (HKEY** hKey, **LPCTSTR** lpSubKey, **PHKEY** phkResult)* - Функція відкриває розділ реєстру.

- ***hKey*** Опис відкриває розділ, що може бути отриманий функціями RegCreateKeyEx й RegOpenKeyEx. Microsoft спростила життя розроблювачеві, визначивши ***стандартні описи***:

  HKEY_CLASSES_ROOT
  HKEY_CURRENT_CONFIG
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
  HKEY_USERS
  Для Windows Me/98/95 також:
  HKEY_DYN_DATA
- ***lpSubKey*** Покажчик на рядок, що завершується нульовим байтом, що містить ***ім'я відкриває*** розділ. Цей розділ повинен бути підрозділом, идентифікуемого описом розділу. Якщо цей параметр NULL, то функція поверне опис самого розділу, тобто роздягнула, ідентифікуемого описом.
- ***phkResult*** Покажчик на змінну, що одержує опис відкритого розділу.

Якщо відкриття відбулося успішно, функція поверне ERROR_SUCCESS, інакше поверне ненульовий код помилки, певний в Winerror.h

***LONG RegOpenKeyEx(HKEY** hKey, **LPCTSTR** lpSubKey, **DWORD** ulOptions, **REGSAM** samDesired, **PHKEY** phkResult)* - Функція відкриває розділ реєстру.

- ***hKey*** Опис відкриває розділ, що може бути отриманий функціями RegCreateKeyEx й RegOpenKey. Діють стандартні описи, перераховані вище.
- ***lpSubKey*** Покажчик на рядок, що завершується нульовим байтом, що містить ім'я відкриває розділ, що. Цей розділ повинен бути підрозділом, ідентифікуемого описом роздягнула. Якщо цей параметр NULL, то функція поверне опис самого розділу, тобто роздягнула, ідентифікуемого описом.
- ***ulOptions*** Зарезервовано - 0.
- ***samDesired*** Визначає права доступу (дії, які буде пророблятиз розділом програміст). Як уже згадувалося, розділ реєстру є системним об'єктом, а отже він має дескриптор захисту, саме в ньому перераховуються права користувачів на об'єкт. Визначено наступні стандартні макроси:

KEY_ALL_ACCESS	Дозволяються будь-які дії над розділом
KEY_ENUMERATE_SUB_KEYS	Дозволяється перерахування підрозділів даного розділу
KEY_READ	Дозволяється читання розділу
KEY_SET_VALUE	Дозволяється створювати, видаляти параметр або встановлювати його значення
KEY_QUERY_VALUE	Дозволяється запит параметра роздягнула

-

- ***phkResult*** Покажчик на змінну, що одержує опис відкритого розділу.

Якщо відкриття відбулося успішно, функція поверне ERROR_SUCCESS, у противному випадку поверне ненульовий код помилки, певний в Winerror.h

LONG RegCreateKeyEx(HKEY *hKey,* ***LPCTSTR*** *lpSubKey,* ***DWORD*** *Reserved,*
LPTSTR *lpClass,* ***DWORD*** *dwOptions,* ***REGSAM*** *samDesired,*
LPSECURITY_ATTRIBUTES *lpSecurityAttributes,* ***PHKEY*** *phkResult,*
LPDWORD *lpdwDisposition)-* Функція створює розділ реєстру. Якщо розділ уже існує, функція відкриває його. Імена розділів не чутливі до регістра.

- ***hKey*** Опис відкриває розділ, що.
- ***lpSubKey*** Покажчик на рядок, що завершується нульовим байтом, що містить ім'я створюваного або відкриваємого розділ, що.
- ***Reserved*** Зарезервовано - 0.
- ***lpClass*** Покажчик на рядок, що завершується нульовим байтом, що містить клас цього розділу. Може бути зігнорований і дорівнює NULL. Використається для підключення до вилученого реєстру.
- ***dwOptions*** Параметр може приймати наступні значення.

REG_OPTION_BACKUP_RESTORE	Якщо прапор установлений, функція ігнорує параметр samDesired і намагається відкрити розділ із правами, для резервного копіювання й відновлення роздягнула. Якщо потік викликає функцію, маючи привілей SE_BACKUP_NAME, то розділ відкривається із правами доступу ACCESS_SYSTEM_SECURITY й KEY_READ. Якщо зухвалий потік має привілей SE_RESTORE_NAME, то розділ відкривається із правами доступу ACCESS_SYSTEM_SECURITY й KEY_WRITE. Якщо потік володіє обома привілеями, то розділ відкривається з комбінованими правами доступу.
REG_OPTION_VOLATILE	Розділ, створений за допомогою цього прапора - мінливий. При цьому інформація зберігається в пам'яті й не зберігається, коли відповідний вулик вивантажується. Для HKEY_LOCAL_MACHINE, це відбувається, коли комп'ютер вимикається або перезавантажується. Для розділів реєстру, завантажених функцією RegLoadKey, це відбувається, коли виконана функція RegUnLoadKey. Функція RegSaveKey не зберігає мінливі розділи реєстру. Цей прапор ігнорується, якщо розділ уже існує.
REG_OPTION_NON_VOLATILE	Розділ, створений за допомогою цього прапора - не змінюємо. При цьому інформація записується у файл реєстру. Функція RegSaveKey зберігає не мінливі розділи.

- ***samDesired*** Визначає права доступу для створюваного розділу.

- ***lpSecurityAttributes*** Покажчик на структуру SECURITY_ATTRIBUTES. Якщо параметр дорівнює NULL, то опис роздягнула не успадковується породженим (дочірнім) процесом. У цю структуру, входить дескриптор захисту роздягнула. Якщо параметр дорівнює NULL, розділ одержує дескриптор захисту за замовчуванням. Список керування доступом (ACL) у заданому за замовчуванням дескрипторі захисту для розділу, успадковуються від батьківського розділу.
- ***phkResult*** Покажчик на змінну, що одержує опис відкритого або створеного розділу.
- ***lpdwDisposition*** Покажчик на змінну, у яку записується одне з наступних значень.

REG_CREATED_NEW_KEY	Розділ не існує й буде створений
REG_OPENED_EXISTING_KEY	Розділ існує

Якщо параметр дорівнює NULL, то ніяка інформація не повертається.
Якщо відкриття відбулося успішно, функція поверне ERROR_SUCCESS, у противному випадку поверне ненульовий код помилки, певний в Winerror.h

Після одержання опису розділу реєстру з ним проробляють потрібні дії, наприклад зчитують або записують значення параметрів. Після проробленних операцій опис роздягнула реєстру повинен бути коректно закритий. Для цього існує функція RegCloseKey, прототип якої показаний нижче.

LONG RegCloseKey(HKEY** hKey**);

- ***hKey*** Опис відкритого розділу, що підлягає закриттю.

Якщо опис успішно звільнений, функція повертає ERROR_SUCCESS, у противному випадку поверне ненульовий код помилки, певний в Winerror.h
Використовуючи вищесказані функції можна написати наступний код, що створює розділ і закриває його.

```
HKEY h;

if(RegCreateKeyEx(HKEY_CURRENT_USER, "12", 0, NULL,
REG_OPTION_VOLATILE,
KEY_WRITE, NULL, &h, NULL)!=ERROR_SUCCESS)
{
printf("Could not create the registry key.");
abort();
}
//робимо певні дії з розділом
RegCloseKey(h);
```

У цьому прикладі створюється розділ для запису з назвою 12, який уже не буде при наступному завантаженні профілю користувача. Потім коректно звільняємо опис створеного розділу.

Після того, як розділ створений або відкритий у ньому можуть бути створені підрозділи, лічена інформація про нього, створені параметри й т.д. Для кожної з перерахованих операцій в Win32 API передбачена відповідна функція.
Розглянемо функцію, що дозволяє одержувати інформацію про розділ реєстру.

LONG RegQueryInfoKey(HKEY** hKey, **LPTSTR** lpClass, **LPDWORD** lpcClass, **LPDWORD** lpReserved, **LPDWORD** lpcSubKeys, **LPDWORD** lpcMaxSubKeyLen, **LPDWORD** lpcMaxClassLen, **LPDWORD** lpcValues, **LPDWORD** lpcMaxValueNameLen, **LPDWORD** lpcMaxValueLen, **LPDWORD** lpcbSecurityDescriptor, **PFILETIME** lpftLastWriteTime**);

- ***hKey*** Опис відкритого розділу. Розділ повинен бути відкритий із правами KEY_QUERY_VALUE.
- ***lpClass*** Покажчик на рядок, що завершується нульовим байтом, що містить клас цього розділу. Може бути зігнорований і дорівнює NULL. Використається для підключення до вилученого реєстру.
- ***lpcClass*** Покажчик на змінну, котра містить розмір буфера, на яку вказує *lpClass*. Розмір повинен включати й завершальний нульовий символ. Після того, як функція поверне значення, у цій змінній буде збережена інформація про розмір буфера-приймача - *lpClass*. Значення, що повертає, не включає нульовий символ. Якщо буфер є недостатнім для збереження даних, то функція повертає значення ERROR_MORE_DATA і змінна містить розмір рядка в байтах, нульовий символ не враховується. Якщо параметр *lpClass* містить правильна адреса, але *lpcClass* є невірним, наприклад NULL, то функція повертає ERROR_INVALID_PARAMETER. В Windows Me/98/95 якщо параметр lpClass містить правильна адреса, але *lpcClass* є невірним, наприклад NULL, те функція повертає ERROR_SUCCESS, замість ERROR_INVALID_PARAMETER.
- ***lpReserved*** Зарезервовано - повинен бути NULL.
- ***lpcSubKeys*** Покажчик на змінну, що одержує в підрозділів даного розділу. Може бути - NULL.
- ***lpcMaxSubKeyLen*** Покажчик на змінну, котра одержує розмір самого довгого імені підрозділу даного розділу, нульовий символ не включається. Може бути NULL. В Windows Me/98/95 розмір включає й нульовий символ.
- ***lpcMaxClassLen*** Покажчик на змінну, котра одержує розмір самого довгого рядка, що визначає клас підрозділу. Розмір, що повертає, не включає нульовий байт.
- ***lpcValues*** Покажчик на змінну, котра одержує число параметрів даного розділу. Може бути - NULL.
- ***lpcMaxValueNameLen*** Покажчик на змінну, що одержує розмір самої довгої назви параметра даного розділу. Розмір не включає нульовий байт. Може бути - NULL.
- ***lpcMaxValueLen*** Покажчик на змінну, що одержує розмір самого довгого значення серед тих, які мають параметри даного розділу. Може бути - NULL.
- ***lpcbSecurityDescriptor*** Покажчик на змінну, котра одержує розмір дескриптора захисту роздягнула, у байтах. Може бути - NULL.
- ***lpftLastWriteTime*** Покажчик на структуру FILETIME, що одержує час останньої модифікації роздягнула. Може бути - NULL.

Функція встановлює члени структури FILETIME відповідно до часу останньої модифікації самого розділу, або параметра, що був змінений пізніше.

Для Windows 9x функція встановлює члени структури FILETIME в 0, тому що система не підтримує механізмів відстеження модифікації розділів реєстру. Якщо функція виконана успішно, повертається ERROR_SUCCESS, у противному випадку повертається ненульовий код помилки, певний в Winerror.h

LONG RegQueryValueEx(HKEY** hKey, **LPCTSTR** lpValueName, **LPDWORD** lpReserved, **LPDWORD** lpType, **LPBYTE** lpData, **LPDWORD** lpcbData**)

Функція повертає інформацію про параметр розділу й значення цього параметра.

- ***hKey*** Опис відкритого розділу. Розділ повинен бути відкритий із правами KEY_QUERY_VALUE.
- ***lpValueName*** Покажчик на З-рядок, що містить назву параметра, про яке виходить інформація. Якщо параметр - NULL або порожній рядок, то повертається інформація про параметр за замовчуванням.

- ***lpReserved*** Зарезервований - NULL.
- ***lpType*** Покажчик на змінну, котра одержує тип даних, збережених у параметрі. Якщо дорівнює NULL, то відповідно, інформація не повертається.
- ***lpData*** Покажчик на масив, що одержує дані параметра. Якщо параметр - NULL, то дані не повертаються. Якщо дані - це рядок, то функція перевіряє наявність нульового символу.
- ***lpcbData*** Покажчик на змінну, котра визначає розмір буфера, що приймає дані з параметра, у байтах. Після того, як функція поверне значення, ця змінна буде містити розмір даних, скопійованих у буфер. Якщо дані носять текстовий характер (REG_xxx_SZ), то також включається й нульовий символ (нульові символи для REG_MULTI_SZ). Якщо розмір буфера, недостатній для збереження даних, то функція поверне ERROR_MORE_DATA і збереже необхідний розмір буфера в змінну, на яку вказує цей параметр. Якщо *lpData* - NULL, а параметр *lpcbData* не нульовий, функція повертає ERROR_SUCCESS і зберігає розмір даних у змінної, на яку вказує *lpcbData*.

Якщо функція виконана успішно, повертається ERROR_SUCCESS, у противному випадку повертається ненульовий код помилки, певний в Winerror.h

Як бачимо, функція *RegQueryValueEx* надає можливість програмістові спочатку перевірити, який розмір даних у параметрі, а потім уже його вважати. Це дозволяє виділяти пам'ять динамічно, по ходу виконання програми. Наприклад, можна написати наступний код, що зчитує дані параметра, якщо вони текстові й виводить їх на екран.

```
HKEY h;
PBYTE pbBuff;
DWORD cBuff=0;
DWORD Type=0;
//Відкриємо розділ
if(RegOpenKeyEx(HKEY_CURRENT_USER,TEXT("asd"),0,
KEY_QUERY_VALUE,&h)==ERROR_SUCCESS)
{
//Визначимо обсяг зчитувальних даних
      if(RegQueryValueEx(h,TEXT("set"),NULL,NULL,NULL,
&cBuff)==ERROR_SUCCESS)
{
if(cBuff>1)
      {
      if((pbBuff=new BYTE [cBuff])==NULL) abort();
      //Зчитуємо інформацію з параметра

      RegQueryValueEx(h,TEXT("set"),NULL,&Type,pbBuff,&cBuff);
            register int i;
            PBYTE tmpBuff;
            if((tmpBuff=new  BYTE [cBuff])==NULL) abort();
            switch(Type)
            {
            case(REG_SZ):
                  cout<<"Type of REG_SZ, data: "<<pbBuff;
                  break;
            case(REG_MULTI_SZ):
                  cout<<"Type of REG_MULTI_SZ, data:\n\t";
                  for(i=0;i<cBuff-1;i++)
```

```
                        pbBuff[i] ? cout<<pbBuff[i] :
                        cout<<'\n'<<'\t';
                        break;
                case(REG_EXPAND_SZ):
                        cout<<"Type of REG_EXPAND_SZ, data:
                        "<<pbBuff<<endl;
                        if(ExpandEnvironmentStrings((PCHAR)pbBuff,
                        (PCHAR)tmpBuff,cBuff)!=0) cout<<tmpBuff;
                        break;
                }

        }
        else cout<<"Value is empty"<<endl;
}
else cerr<<"Error in query"<<endl;
}
else cerr<<"Error in open"<<endl;
```

Наступні дві функції: *RegEnumKeyEx* й *RegEnumValue* використаються для перерахування всіх підрозділів і параметрів, зазначеного описом розділу.

Лекція 9. Захисні механізми програм

Насамперед потрібно знати, що хакер - звичайна людина. Якщо розроблювач вивчить методику його роботи, написання ефективного захисту перестане бути нерозв'язною задачею.

Як правило, хакер добре знає зворотне проектування (reverse engineering), але відносно погано (у порівнянні з розроблювачем) - системне програмування і математику. Крім того, він використовує специфічні інструменти, з якими розроблювач звичайно не знайомий, і працює не з вхідним, а з двійковим кодом. Але це гідний супротивник, що не прощає наївних схем захисту, прорахунків у її реалізації. Загальна стійкість захисту визначається самою слабкою її ланкою. Будь-які хитрування апаратних ключів чи прив'язки до комп'ютера будуть марними, якщо деяка частина алгоритму виконує тривіальну перевірку прапора свій/чужий на відповідність бажаному результату.

Процес протистояння розроблювача і хакера носить динамічний характер, і задача розроблювача - завжди бути на крок попереду.

Кожна нова версія програми повинна мати нову схему захисту, чи сильно модифіковану стару, щоб наявні в хакера наробітки давали осічку.

Звідси випливає ще один важливий принцип: будь-яка схема захисту повинна існувати в єдиному екземплярі. Ні в якому разі не можна застосовувати той самий алгоритм у різних програмах.

Абсолютно надійного захисту не існує. Будь-який захист можна зламати, тому необхідно шукати оптимальне співвідношення витрат на створення захисту і прогнозованих витрат на її злом, з огляду на цінове співвідношення із самим захищаємим ПЗ.

В основному, захист можна розділити на два класи - коли захисні процедури впроваджуються в програмі на етапі її розробки і навісні захисти - коли програма упаковується в т.зв. "конверт" – (спеціальну захисну програму). Зустрічається комбінований метод - в самій програмі є захисні процедури і для того щоб було складніше залізти усередину застосовують зовнішні програми. Практично завжди в цих програмах присутні антівідладочні прийоми. Іноді вони так само упаковують програму, яка захищається. Таких програм безліч: MegaShield, Anti-Lame, Pcrypt, Trap, Gardian Angel, SCRAM!, LockProg, ExeLock, USCC, MSCC, Fds-cp, AdFlt2, XiCOD, UnPackStop, HackStop, CrackStop, ProtEXE, Mask, Exeguard, Mess, XcomOR, Scrypt!, Cn, FFSE, AsPack, AsProtect, Petite, Shrinker, ...

Деякі методи злому захистів.

Будь-яку систему захисту можна розкрити за кінцевий час - це випливає з того, її команди однозначно інтерпретуються процесором. При цьому час, необхідним для розкриття гарної системи захисту, виявляється порівнянним з написанням захищеної програми заново. Однак не всі програми пишуть професіонали, тому часто можна розкрити чи обійти захист, знайшовши її слабкішу ланку, скоротивши час розкриття на кілька порядків. Для цього, зокрема, можна рекомендувати наступні універсальні методи:

- якщо програма захищена тільки від засобів статичного аналізу, вона легко вивчається динамічно, і навпаки;

- "метод зміни одного байта" - у момент, коли система захисту порівнює контрольну інформацію з еталонною, зміною команди переходу вона направляється по правильному шляху;
- аналогічно, результат роботи функції, що повертає поточну контрольну інформацію, може бути підмінений на еталонне (очікуване) значення (наприклад, за допомогою перехоплення відповідного переривання);
- у програмах із захистом, потребуючим ввести ім'я і відповідний йому код можна розібратися в алгоритмі генерації цього коду і написати програму яка буде по введеному імені генерувати правильний код. Також такий підхід можна застосовувати в програмах із прив'язкою до комп'ютера, коли захищена програма по параметрах (дата БИОС, серійний номер твердого диска, його модель, модель материнської плати і процесора, ...) генерує код і просить користувача увести відповідний код.
- коли система захисту розшифрувала критичний код, він може бути скопійований в інше місце пам'яті чи на диск у момент після передачі керування на нього. Окремий випадок - після закінчення роботи програми весь її код розшифрований і доступний.

У професійних системах захисту всі ці прийоми не проходять, і будуть потрібні багато місяців кропіткої роботи в відладнику, щоб розібратися в їхньому функціонуванні. Для дослідження таких систем необхідні спеціальні засоби. Перспективним тут є: інтерпретація програми в цілком віртуальному середовищі, де эмулюються процесор, пам'ять, зовнішні пристрої, операційне середовище і т.д. У цьому випадку описані прийоми протидії виявляються не ефективними; але усе рівно вона відрізняється і програма може це розпізнати. Самим слабким місцем тут буде точна эмуляція тимчасових характеристик апаратури, недокументованих переривань і т.п.

Мабуть, єдиний спосіб надійно захистити свою програму на рівні ідеології - програмний комплекс, експлуатація якого вимагає тісного співробітництва з виробником.

Схема захисту повинна бути розрахована не стільки на виконання контрольних функцій стосовно рядового користувача, скільки на протидію спробам хакера вивести з ладу даний контроль. Тому найпершим етапом в алгоритмі захисту варто виявити хакера, інструменти, якими він користається. Звідси випливає висновок: перед тим, як писати який-небудь захист, розроблювач повинний з'ясувати, чим у даний момент користаються хакери.

Ще раніш добре б довідатися як хакер подивиться на нашу програму. Подивившись код exe файлу дізассемблером - на око можна визначити використану мову і компілятор.

A: Характерні риси :

1) Ассемблер. (Tasm, Masm, Wasm + TLink, WLink, Link). Присутність коду :

```
mov   ax, 3d00h
lea   dx, some01
int   21h
jc    some02
mov   ah,40h
```

Звичайно програми, написані на ассемблері мають дуже мало релокейшенов.

2) Borland Pascal. Присутність в місті входу exe'шника великої купи довгих Call'ов :

```
Call  xxxx:xxxx
Call  xxxx:xxxx
Call  xxxx:xxxx
Call  xxxx:xxxx
Call  xxxx:xxxx
```

Звичайно присутній рядок 'Runtime error at'. Також характерна риса, це те, що дані безладно розкидані по всьому exe'шнику.

3) Turbo C, Turbo C++, Borland C++. Присутність так званого стартап-коду. Майже завжди програма, компілірована цими компиляторами має в крапці входу :

```
mov   dx,xxxx       ; \ При Large-моделі пам'яті
mov   cs:[xxxx], dx  ; /
mov   ah,30h
int   21h
;.... ....;
Push  [xxxx]        ; \ При Large-моделі пам'яті
Push  [xxxx]        ; /
Push  [xxxx]
Push  [xxxx]
Push  [xxxx]
Call  xxxx:xxxx     ; А от це і є виклик процедури _main
                    ; треба копати
```

Інструменти хакера.

Сучасний хакер має у своєму арсеналі набір різноманітних утиліт для злому. Їх можна підрозділити на кілька категорій

А. Відладчик програм. Вони дозволяють: переривати виконання програми при досягненні заздалегідь заданих умов, робити покрокове виконання програми, змінювати вміст пам'яті і регістрів і т.п. Важко сказати, що було першим відладочним чи дебаггером. Усі почалося з TurboDebuggera фірми Borland. Пакет відладки інструментів цієї фірми поставлявся з такими продуктами, як TurboAssembler, TurboPascal, Turbo, Borland C/C++. Захист від дебагування не самоціль, оскільки шифрування програм - не панацея від хакерів. Якщо хакеру, замовляють злом, він має доступ до нормальної копії програми. Тобто він її або може купити, або використовувати її на машині замовника.

SOFTICE, WINICE (просто айс). Цей відладник дотепер є кращим із кращих, і його можливості дозволяють ламати багато захистів. Легко уявити інтерфейс цієї програми по ДОС-ому AFD. Кілька вікон і командний режим роботи. Якщо порівняти з більшістю відладників, то там усі їхні можливості "повішені" на які-небудь клавіші чи пункти меню. В айсі більше можливостей і усі вони реалізовані в "макромову".

Наприклад, серія команд установки точек-останову (брейків):

- BPX –на виконання,
- BPM – на звертання до пам'яті,

- BMSG – на повідомлення Windows,
- BPIO – на звертання до ВУ,
- BPR – на звертання до ділянки пам'яті,
- BPRW – на звертання до модуля,
- BPINT – на переривання.

Плюс ще умови на кожну з команд. Наприклад, треба поставити брейк на ліву кнопку миші на кнопці у вікні.

Даємо команду TASK, вибираємо потрібну задачу. Даємо команду hwnd <ім'я задачі>, вибираємо потрібний handle. Кнопка – це ресурс і дані про неї і її ім'я відоме Windows, а значить і айсу. Отож, вибираєте handle кнопки даємо команду bmsg <хендл>. У Windows багато повідомлень. Набираємо wmsg wm_mouse* і бачимо, що wm_mousefirst=200H. У принципі, якщо ви пом'ятаєте символьне ім'я потрібної вам події, можете відразу його використовувати. Отже, bmsg <хендл> wm_mousefirst. Як ми знаємо у Windows параметри повідомлень заносять в регістри. Отож, якщо вам потрібно можете до будь-якої команди дописати if <регістр>=<вираження> (bpio 21 r if al=1 – перерватися якщо з 21-го порту прочитана 1-ка).

Айс запускається на рівні ядра, тобто їм можна заходити і налагоджувати VXD, DRV. Але не це головне. Такі стародавні штуки, як перехоплення векторів INT1 і INT 3 тепер не проходить.

Висновок: З появою Айса налагодження Windows-програм стало на порядок простіше. І набагато зручніше дизассемблюрування. Принципово змінився стиль деяких атак на захист програм. Тепер не треба крок за кроком дивитися на ассемблер, розбиратися в незначних кодах і захистах. Тепер треба відловити потрібні події і зрозуміти як на нього реагує програма. Але це не завжди буває так просто, як виглядає. Як і раніше, налагодження вимагає знання архітектури ОС. Такий відладник як SOFTICE сильно спрощує підхід до аналізу програм, він не вимагає таких навичок, як дізассемблер. Неважливо наскільки складним був би механізм захисту, але усі зводиться до найпростішої перевірки чи дешифруванню. Злом, у випадку з перевіркою, можна розбити на два етапи:

- Перший: це постановка брейков на "підозрілі" прапори, виявлені в процедурі "захисту".
- Другий: аналіз звертань до "прапорів". По реакції програми можна судити "прапор" це чи просто змінна.

В. Дізассемблери. Роблять дізассемблювання програми для подальшого вивчення отриманого коду. Від дізассемблеру досить легко захиститися - зашифрувати програму. Тоді дизасемблюється тільки архіватор чи кодувальник. Процес перекладу з двійкового виду в символьний, мовою команд якої-небудь мови. Ці речі з'явилися раніш відладчиків, тому що на початку не було архітектури з убудованими засобами відлажування програм. І тим ні менш ці засоби дійшли і до наших днів.

Недоліки:

1. Невірне визначення розмірів даних. Якщо в програмі є цикл із використанням оператора MOV AL, BYTE PTR DS:[BX]. Тоді дізассемблер

зрозуміє, що те, куди звертається оператор можна представити як єдиний блок, наприклад рядок STR DB '0123'. Якщо ж Ви звертаєтеся туди, як це роблять мови високого рівня, то ви одержите:

```
Byte1 db 30h ;'0'
Byte2 db 31h ;'1'
Byte3 db 32h ;'2'
Byte4 db 33h ;'3'
```

Як це може нашкодити? Наприклад, ви дизасемблювали програму закриту HASP ключем. Щоб її зламати, вам потрібно знайти вхід в HASP API. Вона знаходиться відразу за рядком HASPDOSDRV. Ви не знайдете її після дізассемблювання!

2. Відсутність динаміки. Якщо дані в програмі зашифровані, то декомпілятор їх не розшифрує! Величезна кількість незначущих для Вас команд! Неможливість подивитися регістри, стек і пам'ять!

Переваги:

1. Можливість зміни вхідного коду програми.
2. Неможливість виявлення програмою.

Що стосується "неможливості виявлення". Деякі старі відладчики могли пропустити простий фокус. У ДОС, сегменти обмежені 65535, а точніше стековий покажчик SP не може скакати через 0 чи 0FFFFH. Тому якщо ви в програмі зробите SP=0, то відладчики зависнуть. Відладнику потрібний стек, щоб викликався оброблювач одного з відладочних переривань. Якщо стека немає, то поганий захист.

Але розроблювачі дізассемблерів давно врахували складності використання своїх програм. І з'явилися такі програми, як Хакер-VIEW (HIEW) і IDA (Інтерактивний Дізассемблер). У чому їхня перевага? HACKERVIEW випускається як зовнішній переглядач для Нортону. Ви можете переглянути будь-який файл, що виконується, по будь-якому зсуві. Більш того, ви можете "виконати" якусь частину програми чи власну програму, написану на асемблері. Це дозволяє розшифровувати програми й обходити захист від дізассемблювання.

IDA дуже могутній засіб роботи з ассемблерними текстами програм. Має більш зручний інтерфейс. Також дуже добре передбачена архітектура роботи програм у Windows.

Висновок: інтерактивні декомпілятори програм займають свою нішу в інструментарії хакера. В основному це спільне використання з відладчиками, де основну роботу роблять відладчики. Тому іноді простіше в відладчику поставити брейк на потрібну нам подію. Але багато задач не вимагають такого спільного використання.

С. Засоби моніторингу. Це набір утиліт, що відслідковують операції з файлами, реєстром, портами і мережею.

D. Інші утиліти. Їх багато можна знайти на диску типу "Усе для хакера". Це різноманітні редактори, аналізатори.

Найбільш популярні наступні програми моніторингу :
FileMon - утиліта, що дозволяє вести моніторинг всіх операцій з файлами. Вона має зручний фільтр, може зберігати звіт у файлі.
RegMon - аналог FileMon, тільки ведеться моніторинг всіх операцій з реєстром. Аналогічно файлам, тепер безглуздо створювати в реєстрі "секретні" ключі.
PortMon - моніторинг роботи з портами введення/виводу.

Утиліти типу FileMon можуть різко спростити злом програми - легко визначити місце, у якому програма звертається до зазначеного файлу чи ключу реєстру.
TCP_VIEW - монітор з'єднань по TCP/IP

Розглянемо конкретні прийоми захисту насамперед для ДОС-програм

Проблема, що стоїть перед зловмисником при дослідженні захисту від стандартних відладочних засобів - відстеження переривань, що перехоплюються досліджуваною програмою. Практично всі стандартні відладчики для забезпечення своєї нормальної роботи "забирають", як мінімум, 1 і 3 переривання. 1 переривання, називане також трасуючим, використовується для здійснення покрокового режиму роботи відладчику. 3 переривання необхідне для установки міток останову програми по адресах, обумовленим користувачем. Захисний механізм програми обов'язково повинний перехоплювати зазначені переривання для запобігання безперешкодного аналізу захищеної програми відладчиком. Отже, черговою задачею зловмисника на даному етапі є відстеження моменту перехоплення. Якщо ця задача вирішена, то далі він може просуватися по одному з наступних 2-х шляхів.

По-перше, можна заборонити перехоплення переривань шляхом обходу даної ділянки програми. У цьому є частка ризику, тому що підпрограми обробки відповідних переривань можуть виконувати деякі "корисні" функції, необхідні для нормальної роботи програми.

Також можна піти по 2-му шляху. Необов'язково забороняти перехоплення переривань. Якщо ваша кваліфікація досить висока і захисний механізм програми дозволяє це зробити, то можна змінити підпрограми обробки переривань таким чином, щоб після відпрацьовування своїх функцій вони не відразу повертали би керування в основну програму, а передавали б його відповідним підпрограмам відладчика.

Моделювання звертань до ключової дискети.

Якщо не вдається скопіювати ключову дискету, можна спробувати імітувати необхідний формат за допомогою спеціальної програми, що повертає в програму захисту всі ті коди завершення і помилки, як при нормальній роботі. Найчастіше такий метод застосовується коли захист перевіряє наявність якого-небудь ушкодження на поверхні дискети, причому робить це через функції BIOS/INT 13h.

У цьому випадку весь процес теж триває не більш декількох хвилин. Звертання до контролера гнучкого диска через порти можна відстежити, налагоджуючи програму в захищеному режимі. Так само можна імітувати звертання до електронного ключа на LPT чи COM порту.

Зняття програми з пам'яті і методи захисту.

Багато програм знімають начіпні захисти методом зняття з пам'яті (Cup386, Intruder, IceDump, ProcDump). Після того як захист відробив і дав "добро" на виконання, у пам'яті знаходиться вхідна програма в такому ж виді, начебто її запустили звичайним образом, без захистів. Якщо в цей момент зміст ОЗУ записати на диск, то з dump можна витягти первісну програму. У кращому випадку виходить працездатний EXE-файл, практично ідентичний первісній програмі. Хоча іноді зручніше підійти до цих програм так само як до програм з убудованим захистом.

Для розповсюджених типів начіпних захистів існують універсальні програми для їхнього зняття. Майже кожна програма викликає функцію завершення. Якщо неї перехопити, то висока імовірність, що програма в момент завершення в пам'яті не затерта, тобто можна проаналізувати її систему захисту і/чи саму програму.

Эмулюючі відладчики і віртуальні машини.

На додаток до звичайних відладчиків існує особливий клас програм, називаних эмулюючі відладчики. Вони не намагаються коректно виконувати трасування працюючої програми, а самі інтерпретують і виконують її інструкції. Наприклад, замість MOV AX, 56 вони привласнюють змінній Reg_AX - число 56. Існують також відладчики з неповної эмуляцією, що эмулюют тільки "небезпечні" команди, а інші виконують на реальному процесорі.

Такі відладчики нейтралізують практично всі методи протидії відладженню: блокування переривань і пристроїв, роботу з контролерами через порти, підрахунки контрольних сум для виявлення контрольних входів, контроль стеку, а також, методи засновані на особливостях процесора. Плюс до всього, ці відладники мають просто фантастичні можливості для налагодження: можна поставити контрольну крапку по запису/читанню на будь-яку область пам'яті чи ряд портів і т.і.

Методи захисту від відладника.

- Заборона зміни операційного середовища - програма або сама ще раз перенастроює середовище, або взагалі не може працювати в зміненому середовищі.
- Протидія установці контрольних остановів - відладник не може установити контрольний останов, чи програма розпізнає її.
- Порушення інтерфейсу з користувачем приводить до того, що користувач не може прослідити за ходом виконання програми.
- Перехід на свої підпрограми можна здійснювати з оброблювачів переривань.

- Заборона переривань від клавіатури приводить до зависання відладчиків реального режиму.
- Періодично варто перевіряти контрольні суми окремих ділянок програми, з метою уникнути їхньої зміни, наприклад, установки контрольних остановів.
- Використання перепризначення переривань: обмін векторів int 21h і int 10h. Налагодження не запобігає, але сильно заважає.
- Розшифровку захищеної програми через int 08h/int 1Ch. Налагодження не запобігає, але сильно утрудняє.
- Контроль часу (причому заміряти краще не абсолютний час виконання, а відносний час).
- Передача інформації через буфер клавіатури утрудняє налагодження.
- Перевірка дійсної заборони апаратних переривань (пряме перепрограмування контролера переривань).
- Перевірка перехоплених векторів на яке або значення.
- Пропонується проходити ділянка коду з піднятим прапором трасування. При цьому оброблювач int 1h повинний постійно змінювати цей код виходячи зі значень векторів, контрольних сум ділянок програми, і т.д. Крім того, разом з цим може вироблятися розшифровка частин програми, знову ж виходячи з контрольних сум. Хоча б найпростіший контроль часу повинний перешкодити повному трасуванню цього "божевільного коду" вручну.

Методи захисту від дізассемблеру.

Неважливо, наскільки могутньо зробилений захист від відладчику, але цей захист нічого не значить при використанні дізассемблеру. Цілком з'ясовне бажання більшості програмістів працювати з твердою копією досліджуваної програми. Унаслідок цього першочерговою задачею зловмисника при зломі практично будь-якого захисту є дізассемблювання коду програми, що виконується, і одержання лістингу з мнемонічним зображенням ассемблерних команд.

Для захисту від дізассемблерів приходиться використовувати спеціальні методи, що приведені нижче:

1. Модифікація коду програми приводить до того, що дізассемблер не може вірогідно розпізнати інструкції і/чи дані:

2. Зашифровка критичного коду і розшифровка його самою системою захисту перед передачею керування на нього. Таким чином, дешифрування програми відбувається не відразу, а частинами і захист від дізассемблеру виявляється розподіленим за часом. При цьому ніколи не здійснюйте дешифрацію однією підпрограмою, тому що її буде легко обчислити і відключити. Також варто затирати ті ділянки програми, що вже не знадобляться. Шифрування коду програми, що виконується, з метою захисту від дізассемблеру. Шифрування може бути використана лише як частина захисту від дізассемблеру і тому не повинно бути складним;

3. Модифікація коду безпосередньо самою програмою.

4. Приховання команд передачі керування приводить до того, що дізассемблер не може побудувати графа передачі керування наступним шляхом:

- непряма передача керування;
- модифікація адреси переходу в коді програми.

5. Також дізассемблер можна ввести в оману нестандартним форматом модулю, що завантажується, (наприклад, перекрити весь сегмент коду стеком).

5. Перестановка векторів (int 21h -> int 60h). У підсумку в програмі не буде жодного int 21h.

6. Використання для звертання до одній і тій же області пам'яті різних сегментів адрес.

Усі ці прийоми захисту корисні проти дізассемблерів який працюють без взаємодії з користувачем - наприклад Sourcer.

Лекція №10. Захисні механізми програм.

З урахуванням попередньої лекції розглянемо конкретні приклади захисних механізмів:

1. Режим покрокового виконання (трасування) програми ініціюється установкою прапора TF у регістрі прапорів. У покроковому режимі процесор автоматично генерує трассіровочне переривання (INT 1) після виконання кожної команди, якщо перша команда зв'язана з зміною чи пересиланням регістра SS.

Процесори 8086 пропускають трассіровочне переривання після команд зміни чи пересилання будь-яких сегментних регістрів. Ця особливість покрокового режиму роботи називається "втратою трассіровочного переривання" і може бути використана для визначення роботи програми під відладником.

Звичайно процедура обробки трассіровочного переривання використовується програмами налагодження для індикації вмісту регістрів і деяких байтів пам'яті. При обробці переривань процесор зберігає в стеку вміст регістра прапорів, адресу повернення CS:IP. А потім скидає прапори IF і TF, що запобігає покрокове виконання самого оброблювача переривання. Коли процедура обробки переривання завершується, зі стека витягаються колишні стани прапорів і процесор знову переводиться в покроковий режим роботи. Виконуючи програму в режимі трасування, усі популярні відладчики (Turbo Debugger, CodeView, AFD) відслідковують команди PUSHF і INT, і перед їхнім виконанням скидають прапор TF. Цим досягається эмуляція реального режиму трасуруемої програми. При роботі під відладчиком у стек засилаються ті ж самі дані, що й у звичайному режимі роботи.

Наступний фрагмент коду реалізує один з можливих способів реєстрації виконання програми в режимі трасування, що використовує особливості роботи процесора й відладчика в покроковому режимі:

```
        ...
        pop ss          ; у режимі трасування після цієї команди
                        ; переривання int 1 не буде викликане
        pushf
        pop ax          ; одержати прапори в ax
        test ax,0100h   ; чи встановлений прапор TF ?
        jnz tracing
        ...
tracing:                ; робота в покроковому режимі!
        ...
```

Втрата процесором трасуровочного переривання після виконання команди POP SS приводить до того, що відладчик "не помітить" команду PUSHF і в стек буде занесений реальний стан регістра прапорів (з установленим битому TF).

2. Як правило, відладчики дозволяють встановлювати в коді програми контрольні входи (breakpoints). Переривання контрольного входу викликається командою INT 3 з кодом операції 0CCh. Однобайтова команди INT 3 дає можливість установити контрольний вход в будь-яке місце програми, де потрібно перервати нормальне виконання і виконати спеціальні дії.

Оскільки використання переривань 1 і 3 характерно практично для всіх відладочних засобів, можна зробити попередній висновок про роботу програми під відладчиком, якщо вектора цих переривань не вказують на інструкцію IRET (код 0CFh).

Якщо ж у достатній кількості розкидати по коду програми команди виклику переривання контрольного входу, відладчик буде зупинятися на кожній інструкції INT 3, а при нормальному запуску програми її виконання перериватися не буде. Ефективність

цього методу істотно підвищується при використанні команд виклику переривання контрольного входу усередині циклів з великим числом повторень.

Природна реакція хакера в таких випадках - заміна інструкцій INT 3 на NOP. Тому запропонований спосіб бажано доповнювати підрахунком контрольної суми ділянок коду, у яких відбувається виклик INT 3, або доручати оброблювачу переривання контрольної точки корисну роботу. У найпростішому випадку можна просто переустановити вектора цих переривань на процедури, що викликають завершення виконуваної програми. Наприклад, на оброблювачі переривань INT 22 (Program Termination) чи INT 0 (Divide Error).

Непоганих результатів у плані захисту коду від трасування можна домогтися, використовуючи оброблювачі відладочних переривань для динамічної модифікації коду програми.

У наступному прикладі оброблювач переривання INT 1 заміщує пари команд NOP викликом переривання 21h, скидаючи перед поверненням прапор TF для виходу з покрокового режиму. При трасуванні цієї програми відладником CodeView і AFD наш оброблювач INT 1 так і не викликається. Turbo Debugger, хоча й хоче замінити NOP на INT 21, далі виконання першої команди PUSHF йти не бажає. В усіх випадках результат один: вивід на екран повідомлення блокується.

```
code segment
assume cs:code,ds:code,ss:code
org 100h
   start: mov ax,2501h     ; установити оброблювач int 1
   mov dx,offset int1
   int 21h
   pushf                ; ініціювати покрокове
   pop ax               ; виконання програми
   or ah,1              ; TF=1
   push ax
   popf
   mov dx,offset msg       ; вивід повідомлення
   mov ah,09h
print:
   nop                  ; це місце для int 21h
   nop
   int 20h              ; вихід

int1:
   push bp              ; оброблювач int 1
   mov bp,sp
   push ax

   mov ax,[bp+6]          ; скинути прапор TF
   and ah,not 1
   mov [bp+6],ax


   mov word ptr print,021CDh   ; сформувати команду int 21h

   pop ax
   pop bp
   iret
```

```
    msg db 'Normal execution! ',0Ah,0Dh,'$'

code ends
end start
```

3. Ще один спосіб спантеличити практично будь-який відладник - призначення стека в область кодів, що виконуються. Як правило, всі відладники використовують стек трасуемої програми і для своїх потреб, затираючи при цьому відлагоджуємий код. У наступному прикладі серія команд push ax затирає команди виходу з програми, що дозволяє програмі продовжити виконання і після мітки stacktop. При спробі трасування цього фрагмента затираються і самі команди push ax.

```
    ...

    push cs             ; настроїти стек в область коду
    pop ss
    mov sp,offset stacktop
    mov ax,9090h ; nop,nop
    push ax             ; затираємо інструкції
    push ax
    push ax
    nop
    db 0B8h,00h,04Ch    ; mov ax,4C00h
    db 0CDh,21h         ; int 21h
    db 90h              ; nop
stacktop:
    ...             ; тут потрібно відновити покажчики стека!
```

4. Цікаві методи, що дозволяють змінити логіку роботи програми при її покроковому виконанні під відладником, ґрунтуються на використанні конвеєрного принципу виконання команд центральним процесором (черги команд).

Черга команд являє собою набір байтових регістрів у схемі шинного інтерфейсу процесора, у які надходять коди, обрані з програмної пам'яті безпосередньо перед їх виконанням. Коли операційний пристрій процесора зайнятий виконанням команди, шинний інтерфейс самостійно ініціює випереджальну вибірку кодів чергових команд із пам'яті, що дозволяє поєднати в часі фази вибірки і виконання команд. При виконанні операційним пристроєм команд передачі керування (умовні і безумовні переходи, ви клики підпрограм і переривань, повернення з підпрограм і переривань), шинний інтерфейс скидає чергу і починає вибірку команд по новій адресі. Відладчик, виконуючи програму в покроковому режимі, очищає чергу команд на кожнім кроці, викликаючи трасировочне переривання.

Механізми протидії трасуванню можуть використовувати цю особливість. Нехай, наприклад, cmd1 і cmd2 – дві послідовні команди, обрані шинним інтерфейсом у чергу для виконання, причому cmd1 не є командою передачі чи керування командою пересилання/зміни сегментних регістрів. Якщо команда cmd1 змінить "образ" команди cmd2 у пам'яті на cmd2, це ніяк не відіб'ється на ході виконання програми в нормальному режимі, оскільки процесор перейде до виконання наступної команди з черги (cmd2). У покроковому режимі після виконання команди cmd1 відбудеться виклик переривання. При поверненні з переривання черга команд скидається і з пам'яті вибирається модифікована команда cmd2', що змінює логіку роботи програми. Наступний фрагмент коду ілюструє приведені міркування.

```
    ...
```

```
    mov byte ptr _cmd2,0F9h ; 0F9h - код операції stc
_cmd2:
    clc
    jc tracing
    ...       ; код "нормального" режиму
    ...       ; виконання програми
tracing:
    ...        ; робота під відладником!
```

У приведеному прикладі команда mov byte ptr _cmd2,0F9h (cmd1) здійснює заміну наступної команди clc (cmd2) на stc (cmd2'). Однак, у нормальному режимі роботи, замість команди stc буде виконана "стара" команда clc, що до цього моменту вже знаходиться в черзі команд. При виконанні цього фрагмента в покроковому режимі черга команд буде скинута і виконається команда stc, що викликає перехід по мітці tracing (на відповідний оброблювач особливої ситуації).

5. Наступний спосіб виявлення відладчика враховує особливості ініціалізації регістрів програми при завантаженні. Після завантаження програми типу COM, регістри CS,DS,ES і DS містять сегментну адресу PSP. Регістр IP дорівнює 100h, а покажчик стека адресує кінець програмного сегмента (звичайно SP=0FFFEh, але може бути і менше, якщо програмі недоступний повний сегмент пам'яті). У регістр CX заноситься довжина образа програми, рівна довжині COM-файлу. При завантаженні EXE-програм, на PSP указують регістри DS і ES. Початковий стан регістрів CS,IP,SS,SP встановлюється у відповідності зі значеннями в заголовку EXE-файлу. У регістр CX міститься величина, обумовлена наступною формулою:

CX = (PageCnt*200h - HdrSize*10h + 200h - PartPag) % 0FFFFh,

де PageCnt -- довжина образа програми в 512-байтних сторінках (2 байти по зсуві +4 у заголовку EXE-файлу), HdrSize -- довжина заголовка в 16-байтных параграфах (2 байти по зсуві +8), PartPag -- довжина останньої сторінки в байтах (2 байти, зсув +2), % -- операція взяття залишку від розподілу. Як EXE-, так і COM-програми ініцціализирують регістр AX одним з чотирьох значень:

00FFh, якщо 1-й аргумент програми починається символами X,
де X відповідає букві неіснуючого дисковода;
FF00h, якщо 2-й аргумент програми починається символами X,
де X відповідає букві неіснуючого дисковода;
FFFFh, якщо 1-й і 2-й аргументи програми посилаються на
неіснуючі дисководи;
0000h, якщо 1-й і 2-й аргументи не посилаються на неіснуючі дисководи.

При першому завантаженні програми відладчики Turbo Debugger і CodeView завжди обнуляють регістри AX, BX, CX, DX, SI, DI, BP. Після повторного завантаження Turbo Debugger містить у регістрах сміття, що залишилося після першого прогону програми. Таким чином, мінусом відладчика можна вважати регістр CX, у якому при старті програми завжди знаходиться не те, що потрібно. Крім того, Turbo Debugger з великою імовірністю можна "піймати" і на повторному завантаженні, перевіряючи значення в регістрі AX. Відладчиком AFD при завантаженні програми також обнуляються регістри AX, BX, DX, SI, DI, BP, але в регістр CX міститься коректне значення. Після повторного завантаження CX знову встановлюється правильно, а інші регістри містять сміття.

6. За допомогою наступного трюку можна сховати від деяких "розумних" відгадчиків (типу TD.EXE) реально виконувані команди. Досить зробити маскуему команду частиною більш довгої інструкції і передавати на неї керування яким-небудь способом (jmp-ом).

Відладник дизасемблює код, утративши мнемоніку виконуваної в дійсності команди, а недосвідчений хакер, можливо, буде оманений. У приведеному фрагменті створюється ілюзія запису значення E3FFh по адресі 0000h:046Ch (лічильник тиків системного таймера), виконується перехід на мітку addr1.

```
    mov bx,offset addr1
    jmp hidden   ; перехід у середину інструкції
    xor ax,ax
    mov ds,ax
    db 0C7h,06h,6Ch,04h     ; тут Turbo Debugger бачить
                    ; mov word ptr [046C],E3FF
hidden:
    jmp bx   ; перехід на addr1
addr0:
    ...
addr1:
    ...
```

При програванні цього фрагмента під Turbo Debugger'ом інструкція jmp bx сховається в тілі команди mov word ptr [046Ch], 0E3FFh. Після виконання команди jmp hidden, Turbo Debugger спочатку стрибає на інструкцію з адреси addr0, після чого виконується схована команда jmp bx і відбувається перехід по мітці addr1. При цьому команда jmp bx так і залишиться не дизассемблірованою.

Прийоми робота під Windows

Найпростіший спосіб + виклик функції EnumWindows, що перелічує створені в системі вікна верхнього рівня. Одержавши їхні дескриптори, програміст може ідентифікувати запущені процеси (по заголовках вікон, назвам модулів, що виконуються, інформації про версії файлів). Можна також скористатися відладочною по своєму призначенню бібліотекою ToolHelp.

У Windows NT засобів для контролю за станом системи більше, ніж у Windows 9x (наприклад, там мається спеціальна функція IsDebuggerPresent для виявлення відладчика).

Популярний спосіб виявлення відладника - це вимір часу виконання критичних ділянок коду, що дозволяє виявити покроковий режим роботи. Дуже корисно також перевірити, чи не поставлені крапки о станова (байт 0xCC) на деякі функції API чи самої програми. Таким шляхом хакери намагаються визначити, у який момент створюється вікно реєстрації чи перевіряються дані захисту. У лістінгу показана перевірка на наявність точку останову функції CreateWindowEx.

Перевірка наявності крапки о станову у функції CreateWindowEx

```
    extrn  @CheckCRC$qv: FAR
extrn  CreateWindowEx: FAR
public checkdeb             ; прототип - extern LCL int WINAPI checkdeb(void);
checkdeb proc near
push   esi
push   ds
push   cs
pop    ds
  ;  lea     esi, @CheckCRC$qv; у випадку прикладних
  ; функцій Си++ необхідно враховувати перекручування імен
  ; (name mangling)
lea    esi, CreateWindowEx ; функція, імпортована з DLL
```

```
mov    esi,[esi+2]        ; для функції з DLL спершу
  ; знаходимо її адресу в таблиці переадресації,
  ; цей крок варто опустити у випадку контролю
  ; внутрішньої функції
mov    eax,[esi]          ; беремо перші байти функції
and    eax,0FFh           ; найперший
cmp    eax,0CCh           ; є точка останова?
mov    eax,100h
je     Debug_here
xor    eax,eax
Debug_here:
pop    ds
pop    esi
ret                   ; якщо все чисто, повертаємо 0
checkdeb endp
```

Іноді цікаву інформацію можна одержати, вивчивши системне оточення за допомогою функції GetEnvironmentStrings.

Якщо виявлений відладчик чи програма моніторингу, то встає питання про вибір схеми протидії. Активна схема припускає, що необхідно спробувати завісити ворожу програму чи всю систему цілком. Можна, наприклад, злегка зіпсувати стек чи просто викликати функцію DestroyWindow для вікна відладника. Більш екзотичний спосіб порушення штатної роботи небажаної програми: завісити NT використовуючи тільки Win32 API.

Програму легко написати з використанням лише пріоритету HIGH, на який ніяких прав не потрібно. Справа в тім, що планувальник OS нічого не знає про процеси, він розподіляє час між потоками. З цієї причини якщо в однієї програми потоків буде на порядок більше чим в іншої, те всі процеси крім цього фактично не одержать час. Реалізація цієї ідеї:

```
#include <windows.h>
- - - - - - - -
DWORD ThreadProc(LPDWORD)
{
 while(TRUE);
 return 0;
}

int WINAPI WinMain(HINSTANCE hInst,HINSTANCE hPrev,
LPSTR szLine,int Cmd)
{
 DWORD dwResult;
 SetPriorityClass(GetCurrentProcess(),HIGH_PRIORITY_CLASS);
 SetThreadPriority(GetCurrentThread(),THREAD_PRIORITY_HIGHEST);
 while(TRUE)
  {
   CreatThread(NULL,0,(LPTHREAD_START_ROUTINE)ThreadProc,
NULL,0,&dwResult);
  }
 return 0;
}
```

Але активна схема протидії не позбавлена недоліків. По-перше, відладчики і монітори запускаються не тільки в злочинних цілях. По-друге, виконуючи активні дії, захист виявляє себе і дає можливість локалізувати опір.

Більш хитрої і, очевидно, діючої є пасивна схема, коли захист при хакерській атаці просто виключається (переводиться в сплячий режим). У результаті у хакера складається враження, що програма вже зламана чи взагалі не містить захисту. Але коштує неї запустити на чистому комп'ютері, як захист просипається.

Основи побудови захисту

Введення реєстраційного коду чи пароля є відповідальною справою - хакер постарається знайти адресу пам'яті, у який записаний пароль. Потім на звертання по цій адресі ставиться точка останову (команда BPM у SoftICE), що дозволяє знайти початок процедури перевірки реєстраційного коду. Якщо для вводу використовуються стандартні елементи Windows, то алгоритм дій хакера можна формалізувати і виглядає він приблизно так:

1. Установлюєм крапку останову на зчитування тексту зі стандартного елемента вводу (функція GetWindowText, модуля KERNEL32)

2. При виклику цієї функції аналізуємо її параметри й у такий спосіб визначаємо, по якій адресі буде розміщене значення, що зчитується. Ставимо при звертанні до цієї області пам'яті точку останову.

3.При спрацьовуванні цієї точки останову ми попадаємо в аналізатор уведеного значення

Є різновид перевірки серійного номера - це введення імені власника й обчислення еталонного значення. Також це може обчислюватися із серійного номера НЖД, з версії BIOS і т.і. Не так багато варіантів. Але саме головне! ПРИ НАЛАГОДЖЕННІ МИ НЕ ШУКАЄМО ПРАВИЛЬНИЙ ПАРОЛЬ, МИ ШУКАЄМО ПОДІЮ ЗВЕРЕННЯ З УВЕДЕНИМ ПАРОЛЕМ! А як ви там обчислюєте еталон, не важливо.

Розглянемо кілька рішень, що можуть утруднити взлом на цьому етапі.

Порада 0. Намагайтеся якнайменше застосовувати стандартні функції (особливо API) і компоненти VCL. Так що Assembler. Сутність цієї ради сподіваюся очевидна - сучасні дізассемблери вміють розпізнавати стандартні процедури высокорівневих мов. API - взагалі окрема розмова - SoftICE має можливість - завантажувати символьні імена для будь-яких зазначених бібліотек (особливо для KERNEL32.DLL). Налагодження різко спрощується, тому що ми бачимо імена викликуваних функцій і можемо ставити точки останову на виклик функцій по їх імені.

Порада1. Застосовуйте нестандартний спосіб уведення пароля. Найпростіший шлях - написати свій візуальний компонент для введення реєстраційного коду. Він звичайно повинний буде обробляти події від клавіатури, але момент зчитування коду не можна піймати звичайними методами. Це вже щось, але можна шукати введений код в пам'яті. Для цього в SoftICE є зручна команда "S стартова адреса L довжина 'зразок'" , що дозволяє знайти введене значення в пам'яті.

Порада2. Не зберігаєте введений код в одному місці.

Порада 3. Не зберігаєте введений код відкритим текстом. Для початку необхідно завести в програмі 5-10 змінних типу STRING і після введення коду переписати введене значення в них. Робити це найкраще не в одному місці, а розподілити по програмі. У такий спосіб пошук дасть купу адрес, по яких буде знаходитися введений код. Крім того роблять так - по таймері створюють у динамічній пам'яті нову строкову змінну, пишуть у неї код. Потім на наступному спрацьовуванні таймера створюють нову змінну, переписують у неї код, а стару знищують. При навичці можна заполонити пам'ять значеннями введеного коду і зробити пошук майже марним. Таке копіювання можна сполучити з перевіркою коду чи эмуляцією цієї перевірки. Поради 3 і 1 можна об'єднати -

створити свій компонент, що дозволить уводити код нестандартним способом з його одночасною шифровкою.

Якщо, код введений і прийняті міри для того, щоб його було непросто знайти. Тепер наступний крок - аналіз.

Порада 4. Ні в якому разі не аналізуйте код відразу після його введення.

Чим далі введення коду від його аналізу, тим краще. Саме розумне - після введення коду подякувати користувача за співробітництво і повідомити, що виконана реєстрація програми. А аналіз коду зробити, наприклад, через 1-2 хвилини в зовсім іншому місці програми. Намагайтеся взагалі уникати перевірок. Використовуйте можливість працювати з даними, як з кодом і навпаки. Наприклад, раніш був такий прийом. Містилася таблиця адрес чи процедур джампів, а програма до введеного пароля додавала константу, одержуючи зсув у таблиці переходів. Цей прийом менш тривіальний, чим порівняння рядків.

Порада 5. Не перевіряйте код тільки в одному місці і не пишіть для перевірки функції. Досить знайти і відключити перевірку, і захист зламаний.

Порада 6. Не перевіряйте пароль одним алгоритмом. Рекомендується розробити 2-3 алгоритму перевірки. Наприклад 1,2 цифри повинні поділятися на 3, а 3-7, накладені по якому-небудь алгоритмі на ім'я користувача, повинні дати в сумі 4. Ці дві перевірки здійснюємо в різних місцях з досить великим тимчасовим різнобій. Зламавши перший метод хакер не буде догадуватися про існування ще декількох, котрі проявляться згодом.

Порада 7. Ні в якому разі не починайте ніяких дій після перевірки. З невідомої причини більшість програм виглядають приблизно так

```
IF NOT(SuperRegCodeCheck) then
 Begin
  ShowMessage('Невірний код, подальша робота неможлива);
  halt;
 end;
```

У прикладі деяка процедура перевіряє код і при розбіжності починає активні дії, що говорять "от де захист !!". Найкращий крок - виждати день - два. Причому всі дії по перевірці якнайдалі віднести від видачі повідомлень і інших дій, що починаються при виявленні неправильного коду.

Порада 8. Використовуйте відволікаючі маневри.

Крім реальних функцій перевірки коду дуже непогано зробити пару зайвих - вони будуть викликатися після введення коду, проводити активні маніпуляції з уведеним значенням, видавати повідомлення про некоректність уведеного коду - відволікати увагу від реальної перевірки. Якщо, використання серійних номерів не гідне для побудови повноцінного захисту від копіювання, то використовуйте цей метод, як пастку для неуважного кракера. Елементарною пасткою може бути умовний перехід, під MessageBox, що повідомляє "Ви завершили реєстрацію!". А потім викликати загадкової ВЕЛИКОЇ процедури.

Порада 9. Не зберігаєте результатів перевірки в змінній і не використовуйте її для явного обмеження функцій незареєстрованної програми.

Класичний приклад порушення цього правила

```
IF NOT(LegalCopy) then
 ShowMessage('Збереження тільки в зареєстрованій версії')
else
 SaveFile;
```

У такий спосіб елементарний аналіз показує, що змінна LegalCopy зберігає результат перевірки і поставка на неї точки останову може виловити саму перевірку. Відредагувавши це значення в пам'яті можна тимчасово зробити копію "зареєстрованої". Злом зводиться до того, що функція перевірки коду урізається до двох команд ассемблера:

```
MOV [адреса LegalCopy], 1
RET
```

Порада 10. (випливає з 9) Не зберігаєте результатів перевірки на диску чи в реєстрі. Типова помилка - з'ясували, що копія зареєстрована і зробити де-небудь мітку. Відловити це досить просто (див. опис REGMON і FILEMON). Найкращий спосіб - зберегти пароль і ім'я користувача в тім виді, у якому він їх увів. Потім, при кожнім запуску програми перевіряти коректність цього коду.

Порада 11. Нічого не перевіряйте відразу при запуску програми чи відразу після зчитування збереженого чи імені коду.

Захист CRC (контрольні суми). Файл, рядок чи блок даних можна захистити контрольною сумою, щоб потім можна було розрахувати і порівняти з еталоном. При порівнянні з еталоном звичайно випливає звістка обережно - див. перші поради.

Порада 12. Захищайте програми і дані контрольними сумами. Це допоможе не тільки від злому, але і захистить програми від вірусу чи впровадження трояна. Застосовуйте шифровку програм і даних. Дуже непогано зжати програму і дані. Розробляючи свій власний архіватор - RAR-у і ZIP-у він конкуренції не складе, але стиснуті їм дані розтиснути дуже непросто. Та й змінити їх проблематично - прийдеться розтиснути, змінити і стиснути.

Вилов покрокового налагодження програми. Існує багато способів, але вони мало прийнятні під Windows. Найпростіший і надійний спосіб - таймер. При роботі програми періодично фіксуємо системний час і розраховуємо час роботи фрагментів коду між ними. І якщо 200-400 команд процесора працюють 2-3 хвилини, то отут є над чим задуматися.

Порада 13. Не визначайте дату і час стандартними способом !! Придумайте що-небудь оригінальне. Захист "обмеження часу роботи" полягає в тому, що програма фіксує момент свого першого запуску і працює встановлений час (звичайно 20-30 днів). Після цього терміну програма відмовляється запускатися. Як перевірити поточну дату нестандартним способом, по даті на файлах чи реєстру, своєму файлі. Весь фокус в іншому - як зафіксувати на комп'ютері дату першого запуску, щоб зничтоження програми і її повторна установка не давали ефекту. Використання "секретних" файлів у системних папках чи зміни в існуючих файлах легко відловити за допомогою FILEMON. Реєстр те ж відпадає через REGMON. Найбільше оригінально (сьогодні) прошити дату в саму програму і постійно обновляти її на своєму сайті автоматично. Таким чином, відлік неявно йде від моменту скачивання програми із сайта. Є отут правда і мінус - після завершення терміну можна повторно скачати цю програму й одержати ще 15-20 днів. Користувачу чи рано пізно набридне скачивати цю програму і він чи відмовиться від її, чи купить.

Порада 14.Не варто зберігати що-небудь секретне в файлах чи реєстрі. Робота з файлами чи реєстром може бути детально запротокольована і проаналізована, і все таємне стане явним.

Порада 15.Не зберігаєте нічого важливого відкритим текстом, особливо повідомлення типу "Це незареєстрованна версія ...", "Уведений пароль не вірний ...". Вони для хакера - як для бика червона ганчірка. Дійсно, знаходимо таке повідомлення, ставимо точку останову на звертання до ділянки пам'яті з цим повідомленням і одержуємо можливість піймати момент видачі цього повідомлення.

Як правило, захист оперує не менш чим двома наборами даних, отриманими з різних джерел. Наприклад, уведені користувачем ім'я і прізвище й отриманий від розроблювача реєстраційний ключ. Щоб визначити, чи легально використовується програма, над цими

наборами виконуються визначені перетворення, і результати потім порівнюються. Зрозуміло, що відповідні фрагменти коду – це головний об'єкт уваги хакера.

По великому рахунку хакер може застосувати при зломі два різних підходи. Або, досліджувавши алгоритми перетворення даних, навчитися самому генерувати реєстраційні ключі, що програма буде сприймати як правильні. Або просто зіпсувати захист і злегка підправити програму, так щоб захист перестав працювати.

Один з можливих способів боротьби зі спробами відтворити алгоритм генерації ключів – створення коду, що самомодифікується. У цьому випадку статичний код, аналізований дізассемблером, не дає необхідних зведень для відновлення алгоритму генерації реєстраційних ключів. Якщо припустити, що ми успішно засікаємо відладчики і не активізуємо захист у їхній присутності, то локалізувати потрібний фрагмент коду буде складно. А оскільки не можна просто модифікувати код, що динамічно формується тільки на стадії виконання програми, відключити захист теж стає чи неможливо.

Цей список кожен розроблювач може продовжувати сам. Багато свіжих ідей можна почерпнути з Internet і телеконференцій. Найкраще, звичайно, мати знайомого хакера, що візьметься поламати вашу програму, щоб виявити слабкі місця захисту. Але така можливість є не в усіх. Іншим залишається лише на свій страх і ризик прийняти умови гри і вступити в інтелектуальне протиборство зі зломщиками.

Лекція 11. Архівація як форма захисту.

Стиск це є форма кодування. Видаляючи з тексту надмірність, стиск сприяє шифруванню й утрудняє пошук шифру статистичним методом.

Розглянемо оборотний стиск, де первісний текст може бути в точності відновлений зі стиснутого стану. Необоротний стиск використовується для цифрового запису аналогових сигналів, таких як людська мова чи малюнки (gif, jpeg, mpeg).

Найбільш відомий простий підхід і алгоритм оборотного стиску інформації - це кодування серій послідовностей (Run Length Encoding - RLE). Даний підхід складається в заміні ланцюжків чи серій повторюваних байтів на один байт і лічильник числа їхніх повторень.

Наприклад:

44 44 44 11 11 11 11 11 01 33 FF 22 22 - вхідна послідовність

03 44/05 11/00 03 01 03 FF/02 22 - стиснута послідовність

Перший результуючий байт указує скількох разів потрібно повторити наступний байт. Якщо перший байт дорівнює 00, то потім йде лічильник, що показує скільки за ним випливає неповторюваних даних.

Дані методи, як правило, досить ефективні, для стиску растрових графічних зображень (BMP, PCX, TIF, GIF), тому що останні містять досить багато довгих серій повторюваних послідовностей байтів. Недостатком методу RLE є досить низький ступінь оборотного стиску.

Одним з добре відомих методів стиску є алгоритм Хаффмана(1952). Однак, наприкінці 70-х років він був витиснутий. Метод АРИФМЕТИЧНОГО КОДУВАННЯ має схожу з кодуванням Хаффмана функцію, але який дає можливість досягти значної переваги в стиску. Метод Зива-Лемпела, що дає ефективний стиск і підхід застосовує, зовсім відмінний від Хаффмана й арифметичного підхід.

Існує два основних способи проведення стиску: статистичний і словниковий. Кращі статистичні методи застосовують арифметичне кодування, кращі словникові - метод Зива-Лемпела.

У статистичному стиску кожному символу привласнюється код, заснований на імовірності його появи в тексті. Высоковірогідні символи одержують короткі коди, і навпаки.

У словниковому методі групи послідовних символів заміняються кодом. Замінена фраза може бути знайдена в деякому "словнику".

Стислені дані - рядок, файл, текст. Передбачається, що вони виробляються джерелом, що постачає компресор символами, які належать деякому алфавіту. Символами на вході можуть бути букви, літери, слова, крапки. Для конкретного рядка коефіцієнт стиску є відношення розміру стиснутого виходу до його первісного розміру. Для його вираження використовуються багато різних одиниць. Звичайно використовуємо біти на символ (біт/символ) - одиницю, незалежну від представлення вхідних даних. Інші одиниці - відсоток стиску, відсоток скорочення та інші коефіцієнти - залежать від представлення даних на вході.

Моделювання й ентропія.

Одним з найбільш важливих положень теорії стиску з'явилася ідея поділу процесу на дві частин:

- кодувальник, безпосередньо виробляючий стиснутий потік бітів, і

- моделювальник, що поставляє йому інформацію.

Ці дві роздільні частини названі кодуванням і моделюванням. Моделювання привласнює імовірності символам, а кодування переводить ці імовірності в послідовність бітов.

Зв'язок між ймовірностями і кодами встановлений в теоремі Шеннона кодування источника. Символ, очікувана імовірність появи якого є **p** представляється **(-log p)** бітами. Тому символ з високою імовірністю кодується декількома бітами, тоді як малоймовірний вимагає багатьох бітов. Можемо одержати очікувану довжину коду за допомогою усереднення всіх можливих символів за формулою:

$$\Sigma \; p(i) \log p(i) .$$

Це значення називається ентропією розподілу імовірності, тому що це міра кількості безладдя в символах.

Задачею моделювання є оцінка імовірності для кожного символу. З цих імовірностей може бути обчислена ентропія. Важливо відзначити, що ентропія є властивість моделі. У даній моделі оцінювана імовірність символу. Він називається кодовим простором, виділюваним символу, і відповідає розподілу інтервалу (0,1) між символами, і чим більше імовірність символу, тим більше такого "простору" відбирається в інших символів.

Модель є набір імовірностей розподілу, по одному на кожен контекст, на підставі якого символ може бути закодований.

Декодувальник повинний мати доступ до тієї ж моделі, що і кодувальник. Для досягнення цього є три способи моделювання:

- статичне,
- напівадаптоване і
- адаптоване.

Статичне моделювання використовує для всіх текстів модель. Вона задається при запуску кодувальниеа, можливо на підставі зразків типу очікуваного тексту. Така ж копія моделі зберігається разом з декодувальником. Недолік полягає в тому, що схема буде давати поганий стиск, коли кодуемый текст не вписується в обрану модель. Тому статичне моделювання використовують тільки тоді, коли важливі в першу чергу швидкість і простота реалізації.

Напівадаптоване моделювання вирішує цю проблему, будуя для кожного тексту свою модель. Вона будується до самого стиску на підставі результатів попереднього перегляду тексту. Перед тим, як кінчене формування стиснутого тексту, модель повинна бути передана раскодувальнику. Незважаючи на додаткові витрати по передачі моделі, ця стратегія в загальному випадку окупається завдяки кращій відповідності моделі тексту.

Адаптоване моделювання позбавлене зв'язаних з цією передачею витрат. Спочатку і кодувальникик, і раскодувальник привласнюють собі деяку порожню модель, як якби символи всі були рівновірогідними. Кодувальник використовує цю модель для стиску чергового символу, а раскодувальник - для його розгортання. Потім вони обоє змінюють свої моделі однаковим шляхом. Наступний символ кодується і дістається на підставі нової моделі, а потім знову змінює модель. Використовувана модель, що не потрібно передавати явно, буде добре відповідати стиснутому тексту.

Важливо, щоб значення ймовірностей, приймаемих моделлю не були б рівні 0. Тому що якщо символи кодуються -log p бітами, то при близькості імовірності до 0, довжина коду прагне до нескінченності. Нульова імовірність має місце, якщо в зразку тексту символ не зустрівся жодного разу - часта ситуація для адаптованих моделей на початковій стадії стиску. Це - проблема нульової імовірності, яку можна вирішити декількома способами. Один підхід полягає в тому, щоб додавати 1 до лічильника кожного символу.

Задача заміщення символу з імовірністю **p** приблизно **-log p** бітами називається кодуванням. Це вузький зміст поняття, а для позначення більш широкого використовують термін "стиск".

Коди, використовувані в стиснутому тексті повинні підкорятися властивостям префікса: код, використаний у стиснутому тексті не може бути префіксом будь-якого іншого коду.

Добре відомим методом кодування є алгоритм Хаффмана. Стискаючи файл по алгоритму Хаффмана перше що ми повинні зробити - це необхідно прочитати файл цілком і підрахувати скількох разів зустрічається кожен символ з розширеного набору ASCII. Якщо ми будемо враховувати всі 256 символів, то для нас не буде різниці в стиску текстового і EXE файлу.

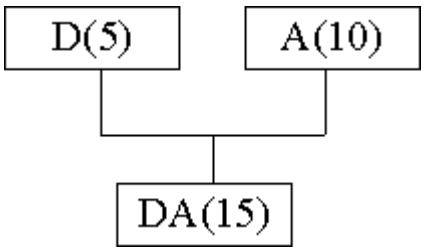
Після підрахунку частоти входження кожного символу, необхідно переглянути таблицю кодів ASCII і сформувати бінарне дерево.

Приклад:

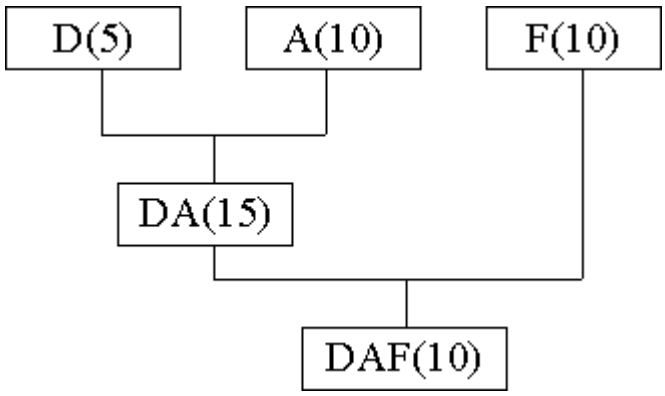
Ми маємо файл довжиною в 100 байт і що має 6 різних символів у собі . Ми підрахували входження кожного із символів у файл і одержали наступне :

A – 10, B – 20, C – 30, D – 5, E – 25, F – 10.

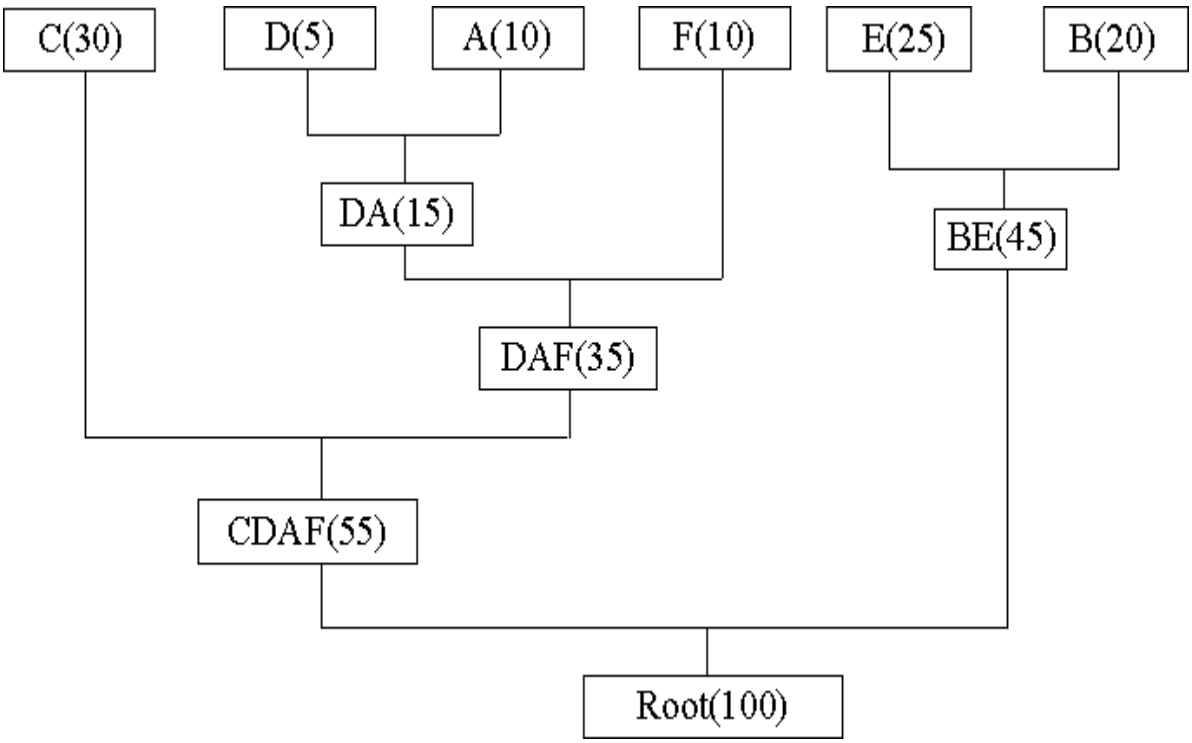
Тепер беремо ці числа і будемо називати їх частотою входження для кожного символу. Візьмемо з останньої таблиці 2 символи з найменшою частотою. У нашому випадку це D (5) і символ F чи A (10). Можна взяти кожної з них, наприклад A. Сформуємо з "вузлів" D і A новий "вузол", частота входження для якого буде дорівнювати сумі частот D і A



Тепер ми знову шукаємо два символи з найнижчими частотами входження. З перегляду D і A і розглядаючи замість них новий "вузол" із сумарною частотою входження. Найнижча частота тепер у F і нового "вузла". Снову зробимо операцію злиття вузлів.



Розглядаємо таблицю знову для наступних двох символів (B і E). Ми продовжуємо цей режим поки всі "дерево" не сформоване, тобто поки усе не зведеться до одного вузла.



Тепер коли наше дерево створене, ми можемо кодувати файл . Ми повинні завжди починати з кореня (Root). Кодуючи перший символ (лист дерева З) Ми прослідкуємо нагору по дереву всі повороти галузей і якщо ми робимо лівий поворот, те запам'ятовуємо 0-й біт, і аналогічно 1-й біт для правого повороту. Так для С, ми будемо йти вліво до 55 (і запам'ятаємо 0), потім знову вліво (0) до самого символу . Код Хаффмана для нашого символу С - 00. Для наступного символу (А) у нас виходить - ліво, право, ліво, ліво , що виливається в послідовність 0100. Виконавши вище сказане для всіх символів одержимо

С = 00 (2 біти)
A = 0100 (4 біти)
D = 0101 (4 біти)
F = 011 (3 біти)
B = 10 (2 біти)
E = 11 (2 біти)

При кодуванні заміняємо символи на дані послідовності.

Алгоритм Хафмана не підходить для адаптованих моделей по двох причинах. По-перше, усякий раз при зміні моделі необхідно змінювати і весь набір кодів. Якщо ж його використовувати в адаптованому кодуванні, то для різних ймовірностей розподілу і відповідних безлічей кодів будуть потрібні свої класи умов для символу. Оскільки моделі можуть мати їх тисячі, то збереження всіх дерев кодів стає надмірно дорогим.

По-друге, метод Хаффмана неприйнятний в адаптованому кодуванні, оскільки виражає значення -log p цілим числом бітов. Це особливо недоречно, коли один символ має високу імовірність. Найменший код, що може бути зроблений методом Хаффмана має 1 біт у довжину. Наприклад, "o" у контексті "to be or not t" можна закодувати в 0.014 біта. Код Хаффмана перевищує необхідний вихід у 71 разів, роблячи точне пророкування марним.

У 70-і роки у алгоритма Хаффмана з'явився конкурент — арифметичне кодування. Цей метод заснований на ідеї перетворення вхідного потоку в одне число з коми. Природно, що чим длинше повідомлення, тим длинше виходить у результаті кодування число. Отже, на виході арифметичного компресора виходить число, менше 1 і більше або рівне 0. З цього числа можна однозначно відновити послідовність символів, з яких воно було побудовано.

Розглянемо роботу арифметичного компресора на прикладі повідомлення “BILL GATES”.

1. Поставимо у відповідність кожному символу повідомлення імовірність його появи в повідомленні (табл. 1).

Символ	Імовірність
Пробіл	1/10
A	1/10
У	1/10
E	1/10
G	1/10
I	1/10
L	2/10
S	1/10
T	1/10

2. Потім привласнимо кожному символу інтервал імовірності в проміжку від 0 до 1. Довжина інтервалу для символу дорівнює імовірності його появи в повідомленні. Розміщення інтервалу імовірності кожного символу не має значення. Важливо, щоб і код ер, і деко дер розташовували символ за однаковими правилами. Інтервали імовірності для символів нашого повідомлення приведені в табл. 2.

Символ	Імовірність	Інтервал
Пробіл	1/10	[0.00,
A	1/10	[0.10,
У	1/10	[0.20,
E	1/10	[0.30,
G	1/10	[0.40,
I	1/10	[0.50,
L	2/10	[0.60,
S	1/10	[0.80,
T	1/10	[0.90,

Таблиця 2 Інтервали ймовірностей для символів повідомлення

У загальному виді алгоритм арифметичного кодування може бути описаний у такий спосіб:

```
Нижняграниця = 0.0;
Верхняграниця= 1.0;
Поки ((Черговийсимвол =ВводЧерговийсимвол()) != КІНЕЦЬ)
    Інтервал = Верхняграниця - НижняГраниця;
            ВерхняГраниця = НижнГраниця +
        [Інтервал*ВерхняГраницяВнтервалуДля(ЧерговийСимвол)];
            Нижняяграниця = НижняГраниця +
        [Інтервал*НижняГраницяІнтервалуДля(ЧерговийСимвол)];
Кінець Поки
Видати (НижняГраница)
```

Для нашого приклада цей алгоритм буде працювати в такий спосіб (табл. 3).

Таблиця 3 Кроки алгоритму арифметичного кодування при обробці повідомлення

ЧерговийСимво	НижняГраница	ВерхняГраниця
	0.0	1.0
У	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
Пробіл	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

Таким чином, згідно з нашою схемою, число 0.2572167752 однозначно кодує повідомлення “BILL GATES”.

Алгоритм арифметичного декодування може бути описаний у такий спосіб:

```
Число = Вводчисла();
Завжди
   Символ =НайтиСимволВІнтервалЯкогоПопадаєЧисло(Число)
   Видати (Символ) Інтервал = ВерхняГраницяІнтервалаДля (Символ)
    -НижняГраницяІнтервалуДля (Символ);
   Число = Число - НижняГраницяИнтервалуДля(Символ);
   Число = Число / Інтервал;
                                        Кінець Завжди
```

У приведеному алгоритмі питання зупинки декодеру - при виявленні спеціального символу EOF (кінець файлу).

Для нашого приклада деко дер буде працювати в такий спосіб (табл. 4).

_	Символ	НижняГра	ВерхняяГр	Інтервал
0.25721677	У	0.2	0.3	0.1
0.57216775	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	Пробіл	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.2	0.3	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.0				
0.8	S	0.8	0.9	0.1

У такий спосіб при обробці числа після визначення символу, в інтервал якого воно попадає.

Головною проблемою в цьому методі є висока точність арифметики для оперування із суцільним бітовим потоком, яким є число, що представляє стиснутий текст.

Найбільш важливими властивостями арифметичного кодування є наступні:

- здатність кодування символу імовірності **p** кількістю бітов довільно близьким до **(-log p)**;
- імовірності символів можуть бути на кожнім кроці різними;
- дуже незначна пам'ять незалежно від кількості класів умов у моделі;
- велика швидкість.

В арифметичному кодуванні символ може відповідати дробовій кількості вихідних бітов. На практиці результат повинний, бути цілим числом бітов, що відбудеться, якщо кілька послідовних високо ймовірних символів кодувати разом, поки у вихідний потік не можна буде додати 1 біт.

Кожен закодований символ вимагає тільки одного целочисельного множення і декількох додавань, для чого звичайно використовується тільки три внутрішніх регістри. Тому, арифметичне кодування ідеальне підходить для адаптованих моделей

Складність арифметичного кодування полягає в тому, що воно працює з імовірністю розподілу, що накопичується. Розглянемо різні моделі.

1. КОНТЕКСТУАЛЬНІ МОДЕЛІ.

1.1 Моделі з фіксованим контекстом.

Статистичний кодувальникик, якимсь є арифметичний, вимагає оцінки розподілу імовірності для кожного кодуемого символу. Найпростіше привласнити кожному символу постійну імовірність, незалежно від його положення в тексті, що створює просту контекстно - вільну модель. Наприклад, в англійській мові імовірності символів ".", "e", "t" і "k" звичайно складають 18%, 10%,

8% і 0.5% відповідно (символ "." використовується для позначення пробілів). Отже в цій моделі дані букви можна закодувати оптимально 2.47,

3.32, 3.64 і 7.62 бітами за допомогою арифметичного кодування. У такій моделі кожен символ буде представлений у середньому 4.5 бітами. Це є значенням ентропії моделі, заснованої на імовірності розподілу букв в англійському тексті. Ця проста статична контекстно - вільна модель часто використовується разом з кодуванням Хаффмана[35].

Імовірності можна оцінювати адаптивне за допомогою масиву лічильників - по одному на кожен символ. Спочатку усі вони встановлюються в 1 (для запобігання проблеми нульової імовірності), а після кодування символу значення відповідного лічильника збільшується на одиницю. Аналогічно, при декодуванні відповідного символу раскодувальник збільшує значення лічильника. Імовірність кожного символу визначається його відносною частотою. Ця проста адаптивна модель незмінно застосовується разом з кодуванням Хаффмана[18,27,32,52,104, 105].

Більш складний шлях обчислення ймовірностей символів лежить через визначення їхньої залежності від попереднього символу. Наприклад, імовірність проходження за буквою "q" букви "u" складає більш 99%, а без обліку попереднього символу - усього 2.4%(2). З урахуванням контексту символ "u" кодується 0.014 біта і 5.38 біта в противному випадку. Імовірність появи букви "h" складає 31%, якщо поточним символом є "t", і 4.2%, якщо він невідомий, тому в першому випадку вона може бути закодована 1.69 битка, а в другому - 4.6 біта. При використанні інформації про попередні символи, середня довжина коду (ентропія) складає 3.6 біта/символ у порівнянні з 4.5 біта/символ у простих моделях.

Цей тип моделей можна узагальнити відносно о попередніх символів, використовуваних для визначення імовірності наступного символу. Це визначає контекстно-обмежену модель ступеня о.

Контекстно-обмежені моделі незмінно застосовуються адаптивно, оскільки вони мають можливість пристосовуватися до особливостей стисливого тексту. Оцінки імовірності в цьому випадку являють собою просто лічильники частот, формовані на основі вже переглянутого тексту.

Спокусливо думати, що модель більшого ступеня завжди досягає кращого стиску. Ми повинні вміти оцінювати імовірності щодо контексту будь-якої довжини, коли кількість ситуацій наростає экспоненціально ступеня моделі.

Т.ч. для обробки великих зразків тексту потрібно багато пам'яті. В адаптивних моделях розмір зразка збільшуються поступово, тому великі контексти стають більш виразними в міру здійснення стиску. Для оптимального вибору - великого контексту при гарному стиску і маленькому контексті при недостатності зразка - випливає робити змішану стратегію, де оцінки ймовірностей, зроблені на підставі контекстів різних довжин, поєднуються в одну загальну імовірність. Існує кілька способів виконувати перемішування.

1.2 Контекстуально - змішані моделі.

Змішані стратегії використовуються шляхом об'єднання оцінок і складаються в присвоєнні ваги кожної моделі й обчисленню зваженої суми ймовірностей. Сума ваг повинна дорівнювати 1. Як вибирати ваги? Один шлях складається в присвоєнні заданої безлічі ваг моделям різних порядків. Іншої, для додання більшої виразності моделям вищих порядків, - в адаптації ваг у міру виконання стиску. Однак, відносна важливість моделей змінюється разом з контекстами і зв'язаними з ними лічильниками.

1.3 Алфавіти.

Принцип контекстно-обмеженого моделювання може бути застосуємо для будь-якого алфавіту. 8-бітовий алфавіт ASCII звичайно добре працює з максимальною довжиною контексту в кілька символів. Якщо звертання відбувається до биток, то можна застосовувати двійковий алфавіт (наприклад, при стиску зображень). Для такого алфавіту існують дуже ефективні алгоритми арифметичного кодування незважаючи на те, що у випадку 8-бітового алфавіту було б закодоване в 8 разів більше двійкових символів. Іншою крайністю може бути розбивка тексту на слова [67]. У цьому випадку необхідні тільки маленькі контексти

Отже - звичайно буває досить одного, двох слів. Керування дуже великим алфавітом являє собою окрему проблему.

1.6. Практичні контекстно-обмежені моделі.

DAFC - одна з перших схем, що змішують моделі різних порядків і адаптують її структур. Вона засновує контексти тільки на найбільше символах, що часто зустрічаються. Звичайно перші 31 символ, лічильники яких досягли значення 50, адаптивна використовуються для формування контекстів 1-го порядку. Застосування контекстів нижчого порядку гарантує, що DAFC працює швидко і використовує при цьому обмежений (і щодо невеликої) обсяг пам'яті.

ADSM. Символи в кожнім контексті класифікуються відповідно до їхніх частот; цей порядок передається за допомогою моделі 0-ої ступеня. Перевагою ADSM є те, що вона може бути реалізована в якості швидкого предпроцессору до системи 0-го порядку.

PPMC - ретельно пристосована Моффатом для поліпшення стиску і збільшення швидкості виконання. Застосовує масштабіровані лічильники з максимальною точністю 8 бітов (придатне для широкого спектра файлів), накладає обмеження на необхідну пам'ять, очищаючи і перебудовуючи модель у випадку вичерпання простору.

WORD є схема подібна PPMC, але алфавіт "слів" - з'єднаних символів алфавіту - і "не слів" - з'єднаних символів, що не входять у цей алфавіт. Первісний текст перекодується для перетворення його у відповідну послідовність слів і не слів [10]. Для них використовуються різні контекстно-обмежені моделі 0-го і 1-го порядків. Слово

оцінюється попередніми словами, не слово - не словами. Для перебування ймовірностей використовується метод В, а через великий розмір алфавіту - ледачі виключення. Застосовуються також і обновлювані виключення. Модель припиняє рости, коли досягає визначеного максимального розміру, після чого статистики змінюються, але нові контексти не додаються.

Якщо зустрічаються нові чи слова не слова, вони повинні визначатися іншим способом. Це здійснюється передачею спочатку довжини (обраної з числі від 0 до 20) з моделі довжин 0-го порядку. Потім знову використовується контекстно-обмежена модель букв (чи неалфавітних символів у випадку не слів). У підсумку завантажуються і змішуються 10 моделей: 5 для слів і 5 для не слів.

З всіх описаних, контекстно-обмежені методи звичайно дають кращий стиск, але можуть бути дуже повільними. Відповідно до будь-якої практичної схеми, час, необхідний на кодування і раскодования росте тільки лінійно щодо довжини тексту. Крім того, воно росте принаймні лінійно до порядку найбільшої моделі. Однак, для ефективності реалізації необхідна особлива увага на деталі. Будь-яка збалансована система буде являти собою складний компроміс між часом, простором і ефективністю стиску.

Кращий стиск досягається на основі дуже великих моделей, що завжди забирають пам'яті більше, ніж самі стисливі дані. Дійсно, основним фактором поліпшення стиску за останні 10 років є можливість доступу до великих обсягів пам'яті, чим раніш. Через адаптацію ця пам'ять відносно дешева для моделей не нужденних у чи підтримці обслуговуванні, тому що вони існують тільки під час власне стиску і їхній не треба передавати.

Структури Даних, придатних для змішаних контекстуальних моделей звичайно ґрунтуються на деревах цифрового пошуку. Контекст представляється у виді шляху вниз по дереву, що складається з вузлів-лічильників. Для швидкого відшукання розташування контексту відносно вже знайденого більш довгого (що буде случатися часто при доступі до моделей різного порядку) можна використовувати зовнішні покажчики.

Це дерево може бути реалізоване через хеш-таблиця, де контекстам відповідають елементи[78]. З колізіями справа мати не обов'язково, оскільки хоча вони й адресують різні контексти, але малоймовірні і на стиск будуть впливати (скоріше на коректність системи).

2. ІНШІ СТАТИСТИЧНІ МОДЕЛІ.

Контекстно-обмежені методи є одними з найбільш відомих і ефективних. Найкращі моделі відбивають процес створення тексту, де символи вибираються не просто на підставі декількох попередніх. Ідеальним буде моделювання думок суб'єкта, що створив текст.

Це було використано Шенноном для перебування межі стиску для англійського тексту. Він працював з людьми, що намагаються угадати наступні друг за другом символи тексту. На підставі результатів цього досвіду, Шеннон уклав, що краща модель має значення ентропії між 0.6 і 1.3 біт /символ.

Розглянемо класи моделей, що пропонують деякий компроміс між контекстно-обмеженими моделями і загадковою міццю процесів людського мислення.

2.1 Моделі станів.

Вірогідні моделі з кінцевим числом станів ґрунтуються на кінцевих автоматах (КА). Вони мають безліч станів S(i) і віроідність переходу P(i,j) моделі зі стану й у стан j. При цьому кожен перехід позначається унікальним символом. Т.ч., через послідовність таких

символів будь-який вихідний текст задає унікальний шлях у моделі (якщо він існує). Моделі з кінцевим числом станів здатні імітувати контекстно-обмежені моделі.

Крім здійснення кращого стиску, моделі з кінцевим числом станів швидше в принципі. Поточне стан може заміщати імовірність розподілу для кодування, а наступне стан простий визначається по дузі переходу. На практиці стану можуть бути виконані у виді зв'язного списку, що вимагає ненабагато більше обчислень.

На жаль задовільні методи для створення гарних моделей з кінцевим числом станів на підставі зразків рядків ще не знайдені.

Один підхід полягає в перегляді всіх моделей можливих для даного числа станів і визначенні найкращої з них. Ця модель росте экспоненціально кількості станів і годить тільки для невеликих текстів.

Більш евристичний підхід складається в побудові великої початкової моделі і наступному скороченні її за рахунок об'єднання однакових станів.

2.2. Маркоскі моделі

Єдиний із працюючих досить швидко, щоб його можна було застосовувати на практиці, метод моделювання з кінцевим числом станів, називається динамічним стиском Маркова (ДМС). ДМС адаптивно працює, починаючи з простої початкової моделі, і додає в міру необхідності нові стани. На жаль, виявляється що вибір евристики і початкової моделі забезпечує створюваної моделі контекстно-обмежений характер, через що можливості моделі з кінцевим числом станів не використовуються в повну силу. Головна перевага ДМС над описаними в розділі 1 моделями складається в пропозиції концептуально іншого підходу, що дає їй можливість при реалізації працювати швидше.

У порівнянні з іншими методами стиску ДМС звичайно здійснює побітове введення, але принципової неможливості символьно-оріентованої версії не існує. Однак, на практиці такі моделі найчастіше вимагають багато ОП, особливо якщо використовується проста СД. Моделі з побітовим уведенням не мають проблем з пошуком наступного стану, оскільки в залежності від значення наступного біта існують тільки два переходи з одного стану в інше. Ще важливо, що працююча з бітами модель на кожнім кроці здійснює оцінку у формі двох ймовірностей p(0) і p(1) (у сумі що дають 0). У цьому випадку застосування адаптивного арифметичного кодування може бути особливо ефективним.

Для старту ДМС потрібна початкова модель. Причому проста, оскільки процес клонування буде змінювати її у відповідності зі специфікою зустрінутої послідовності. Однак, вона повинна могти кодувати всі можливі вхідні послідовності. Найпростішим випадком є модель з 1 станом, показаний на малюнку 3, що є цілком задовільної. При початку клонування вона швидко виростає в складну модель з тисячами станів. Небагато кращий стиск може бути досягнуте для 8-бітового введення при використанні початкової моделі, що представляє 8-бітові послідовності у виді ланцюга, як показано на малюнку 4, чи навіть у виді двійкового дерева з 255 вузлів. Однак, початкова модель не є особливо вирішальної, тому що ДМС швидко пристосовується до вимог кодуемого тексту.

2.3 Моделі новизни.

Вони працюють за принципом, що поява символу у вхідному потоці робить більш ймовірним його нова поява в найближчому майбутньому. Цей механізм аналогічний стопі книг: коли книга необхідна, вона витягається з будь-якого місця стопи, але після використання кладеться на самий верх. Т.ч. найбільш популярні книги будуть ближче до вершини, що дозволяє їх швидше знаходити. Звичайно вхідний потік розбивається на слова (зчеплені символи, розділені пробілом), що використовуються як символи.

Символ кодується своєю позицією в обновлюваному списку (стопці книг). Застосовуються коди перемінної довжини, у якому слова, розташовані ближче до вершини мають більш короткий код. Існує кілька способів організації використовуваних при цьому

списків. Один - переміщати символи в самий початок після їхнього кодування, іншої - переміщати їх убік початку лише на деяку відстань.

3. СЛОВАРНІ МЕТОДИ.

В основі цих методів лежить ідея зовсім відмінна від ідеї статистичного стиску. Словниковий кодувальникик домагається стиску заміною груп послідовних символів індексами деякого словника. Словник є список таких словосполучень, що будуть часто використовуватися. Індекси улаштовані так, що займають менше місця, чим кодуємі ними слова, за рахунок чого і досягається стиск. Цей тип стиску ще відоме як "макро" або "книги кодів". Відмінність між моделюванням і кодуванням для словникових методів важко установити, оскільки при зміні словників коди звичайно не змінюються.

Словникові методи звичайно швидкі, із причин, що один код на виході відповідає декільком вхідним символам і що розмір коду звичайно відповідає машинним словам. Можна показати, що більшість словникових кодувальників можуть бути відтворені за допомогою контекстно-обмежених моделей, тому їх головним достоїнством є не якість стиску, а економія машинних ресурсів.

Центральним рішенням при проектуванні словникової схеми є вибір розміру запису кодового словника. Деякі розроблювачі накладають обмеження на довжину збережених фраз. Наприклад, при кодуванні діадами вони не можуть бути більш двох символів. Вибір фраз може здійснюватися: статичним, напівадаптивним, адаптивним способом.

Найпростіші словникові схеми застосовують статичні словники, що містять тільки короткі фрази. Вони особливо годять для стиску записів файлу, такого, як, наприклад, бібліографічна база даних, де записи повинні декодуватися випадковим образом, але при цьому та сама фраза часто з'являється в різних записах. Однак, адаптивні схеми, що допускають великі фрази, досягають кращого стиску. Розглянуте нижче стиск Зива-Лемпела є загальний клас методів стиску, що відповідають цьому опису і перевершують інші словникові схеми.

3.1 Стратегія розбору.

Раз словник обраний, існує кілька варіантів вибору фраз із вхідного тексту, що заміщаються індексами словника. Метод розбивки тексту на фрази для кодування називається розбором. Найбільш швидкісним підходом є *ретельний* розбір, коли на кожнім кроці кодувальникик шукає в словнику самий довгий рядок, який відповідає поточному рядку тексту. На жаль ретельний розбір не обов'язково буде оптимальним, оскільки межа на те, як далеко вперед повинний дивитися кодувальникик, не встановлена.

Алгоритми оптимального розбору вимагають попереднього перегляду тексту. З цієї причини на практиці широко використовується ретельний метод, навіть якщо він не оптимальний, тому що проводить однопрохідне кодування з обмеженою затримкою.

Компромісом між ретельним і оптимальним розборами є метод розміщення самого довгого фрагмента в початок - LFF. Цей підхід шукає саму довгу підстроку вводу (не обов'язково починаючи спочатку), що також є й у словнику. Ця фраза потім кодується, і алгоритм повторюється доти, поки не закодуються всі підстроки.

Наприклад, для словника M = a,b,c,aa,aaaa,ab,baa,bccb,bccba , де всі рядки кодуються 4-мя бітами, а LFF - розбір рядка "aaabccbaaaa" спочатку установлює "bccba" як самий довгий фрагмент. Остаточний розбір рядка є: "aa,a,bccba,aa,a", і рядок кодується 20-ма бітами. Ретельний розбір дає "aa,ab,c,c,baa,aa" (24 біта), тоді як оптимальний розбір є "aa,a,bccb, aaaa" (16 бітов). У загальному випадку показники стиску і швидкості для LFF знаходяться між ретельним і оптимальним методами. Як і для оптимального стиску LFF вимагає перегляду усього вводу перед ухваленням рішення про розбір.

Статичні словникові кодувальники корисні в тому випадку, якщо достатній невисокий рівень стиску досягається за рахунок невеликих витрат. Запропонований у різних формах швидкий алгоритм кодування діадами підтримує словник розповсюджених пар чи символів діад. На кожнім кроці кодування чергові два символи перевіряються на відповідність діадам у словнику. Якщо воно є, вони разом кодуються, інакше кодується тільки перший символ, після чого покажчик просувається уперед відповідно на дві чи одну позиції.

Природним розвитком статичного n-адного підходу є створення свого словника для кожного кодуемого тексту. При цьому виникає багато рішень, близьких до оптимального, і більшість з них схожі. Вони звичайно починають зі словника, що містить усі символи вихідного алфавіту. Потім додають до них розширені діады, тріади і т.д., поки не заповниться весь словник.

Вибір кодів для елементів словника виявляє собою компроміс між якістю стиску і швидкістю кодування. Вони являють собою рядки бітів цілої довжини, причому кращий стиск будуть забезпечувати коди Хаффмана, одержувані з частот фраз. Однак, у випадку рівної частоти фраз, підхід, що використовує коди перемінної довжини, мало чого може запропонувати. Тому тут більш зручними є коди з фіксованою довжиною. Якщо розмір кодів дорівнює машинним словам, то реалізація буде і швидше, і простіше. Компромісом є дворівнева система. Наприклад, 8 бітів для 1 символьних фраз і 16 бітів - для більш довгих.

Майже всі практичні словникові кодувальники належать класу алгоритмів, що витікає з роботи Зіва і Лемпела. Сутність полягає в тому, що фрази заміняються покажчиком на те місце, де вони в тексті вже раніше з'являлися. Це сімейство алгоритмів називається методом Зива-Лемпела і позначається як LZ-стиск. Цей метод швидко пристосовується до структури тексту і може кодувати короткі функціональні слова, тому що вони дуже часто в ньому з'являються. Нові слова і фрази можуть також формуватися з частин раніше зустрінутих слів.

Раскодування стиснутого тексту здійснюється просто - відбувається проста заміна покажчика готовою фразою зі словника, на яку той указує. На практиці LZ-метод досягає гарного стиску, його важливою властивістю є дуже швидка робота раскодувальника.

Однієї з форм такого покажчика є пари (m,l), що заміняє фразу з l символів, що починається зі зсуву m у вхідному потоці. Наприклад, покажчик (7,2) адресує 7-ий і 8-ий символи вихідного рядка. Використовуючи це позначення, рядок "abbaabbbabab" буде закодована як "abba(1,3)(3,2)(8,3)".

Поширено невірне представлення, що за поняттям LZ-методу стоїть єдиний алгоритм. Просування покажчика в раніше переглянуту частину тексту може бути необмеженим (расширене вікно) чи обмежене вікном постійного розміру з N попередніх символів. N звичайно складає кілька тисяч. Обрані підстроки також можуть бути необмежені чи обмежені безліччю фраз, обраних відповідно до деякого задуму.

Кожна комбінація цих умов є компромісом між швидкістю виконання, обсягом необхідної ОП і якістю стиску. Вікно, що розширюється, пропонує кращий стиск за рахунок організації доступу до більшої кількості підстрок. Якщо пам'яті для вікна буде не вистачати, відбудеться скидання процесу, що також погіршить стиск до нового збільшення вікна. Вікно постійного розміру позбавлено цих проблем, але містить менше підстрок, доступних покажчику.

Таблиця 2 містить зведення про основні відмінності в різних реалізаціях цього методу.

Таблиця 2. Основні варіанти LZ-схеми.

Ім'я	Автори	Відмінності

LZ77	Ziv and Lempel [1977]	Покажчики і символи чергуються. Покажчики адресують підстроку серед попередніх N символів.
LZR	Roden et al. [1981]	Покажчики і символи чергуються. Покажчики адресують підстроку серед усіх попередніх символів.
LZSS	Bell [1986]	Покажчики і символи розрізняються бітом. Покажчики адресують підстроку серед попередніх N символів.
LZH	Brent [1987]	Аналогічно LZSS, але на другому кроці для покажчиків застосовується кодування Хаффмана.
LZW	Welch [1984]	Вивід містить тільки покажчики. Покажчики адресують раніше розібрану підстроку. Покажчики мають фіксовану довжину.

4. ПОРІВНЯННЯ

Порівняння двох схем стиску не складається в простому визначенні, яка з них краще стискає. Навіть ідучи убік від умов для який виміряється стиск - вид тексту, питання адаптації і багатосторонності в роботі з різними жанрами - існує багато інших факторів, що враховуються, таких як необхідна кількість пам'яті і часу для здійснення стиску. Задача оцінки є складений, оскільки ці фактори повинні бути розглянуті і для кодувальниеа, і для декодуровальника, що можуть залежати від типу стисливого тексту.

Майже всі методи мають параметри, що звичайно впливають на швидкість роботи і необхідні обсяги пам'яті.

Малюнок 7 характеризує зразки текстів, що оброблялися вищевказаними методами. Вони включають книги, статті, чорно-білі зображення та інші види файлів, розповсюджені в системах ЕОМ.

Таблиця 4 містить отримані результати, виражені в бітах на символ.

Text	Type	Format	Content	Size
Bib	bibliography	UNIX "refer" format, ASCII	725 references for books and papers on Computer Science	111.261 characters
news	electronic news	USENET batch file	A variety of topics	377.109 characters
Obj2	object code	Executable file For Apple Macintosh	"Knowledge Support System" program	246.814 characters

progc	Program	Source code in "C", ASCII	Unix utility "compress" version 4.0	39.611 characters
progp	Program	Source code in Pascal, ASCII	Program to evaluate compression performance of PPM	49.379 characters

Таблиця 4. Результати досвідів по стиску (біти на символ)

Текст	Розмір	LZB	LZFG	HUFF	DAFC	ADSM	PPMC	WORD
Bib	111261	3.17	2.90	5.24	3.84	3.87	*2.11	2.19
news	377109	3.55	3.44	5.23	4.35	4.35	*2.65	3.08
obj1	21504	4.26	4.03	6.06	5.16	5.00	*3.76	4.50
pic	513216	1.01	*0.87	1.66	0.90	1.03	1.09	0.89
progc	39611	3.08	2.89	5.26	4.43	4.20	*2.49	2.71
У середньому	224402	3.18	2.95	4.99	4.02	3.95	*2.48	2.76

Досвід показує, що більш витончені статистичні моделі досягають кращого стиску. Гіршу характеристику мають найпростіші схеми - діади і кодування Хаффмана.

6. ПОДАЛЬШІ ДОСЛІДЖЕННЯ.

Існують 3 напрямку досліджень у даній області:

- підвищення ефективності стиску,
- прискорення роботи алгоритму
- здійснення стиску на підставі нової системи констекстів.

Лекція 12. Основні поняття криптографії.

Наука, що займається питаннями безпечного зв'язку за допомогою зашифрованого представлення і передачі інформації називається к***риптологією*** (kryptos - таємний, logos - наука). Вона у свою чергу розділяється на два напрямки *криптографію* і *криптоаналіз.*

Криптографія - наука про створення стійких (до злому) шифрів і створенню безпечних методів зв'язку. Вона займається пошуком математичних методів перетворення інформації.

Криптоаналіз - даний розділ присвячений дослідженню можливості читання повідомлень без знання ключів, тобто зв'язаний безпосередньо зі зломом шифрів. Люди, що займаються кріптоаналізом і дослідженням шифрів називаються *кріптоаналітиками*.

Шифр - сукупність оборотних перетворень безлічі відкритих текстів (вхідного повідомлення) на безліч зашифрованих текстів, проведених з метою їхнього захисту. Конкретний вид перетворення визначається за допомогою ключа шифрування.

Зашифрування - процес застосування шифру до відкритого тексту.

Розшифрування - процес зворотнього застосування шифру до зашифрованого тексту.

Дешифрування - спроба прочитати зашифрований текст без знання ключа, тобто злом шифротексту чи шифру.

Тут варто підкреслити різницю між расшифруванням і дешифруванням. Перша дія проводиться законним користувачем, що знає ключ, а друга - кріптоаналітиком чи могутнім хакером.

Криптографічна система - сімейство перетворень шифру і сукупність ключів. Сам по собі опис алгоритму не є криптосистемой. Тільки доповнене схемами розподілу і керування ключами він стає системою. Як правило, опис алгоритму шифрування вже містить у собі всі необхідні частини.

Одне з основних правил криптографії (якщо розглядати її комерційне застосування) можна виразити в такий спосіб: *злом шифру з метою прочитати закриту інформацію повинний обійтися зловмиснику набагато дорожче, ніж ця інформація коштує насправді*.

Сучасні криптосистеми класифікують у такий спосіб:

- Симетричні (із закритим, секретним ключем),
- Асиметричні (з відкритим ключем)

Криптосистеми можуть забезпечувати не тільки таємність переданих повідомлень, але і їхню аутентичність (дійсність), а також підтвердження дійсності користувача.

Розглянемо ці криптосистеми.

Симетричні криптосистеми (із секретним ключем - secret key systems). Дані криптосистеми побудовані на основі збереження в таємниці ключа шифрування. Процеси за шифрування і розшифрування використовують один ключ. Таємність ключа є постулатом. Основна проблема при застосуванні симетричних криптосистем для зв'язку полягає в складності передачі обом сторонам секретного ключа. Однак дані системи мають високу швидкодію. Розкриття ключа зловмисником грозить розкриттям тільки тієї інформації, що була зашифрована на цьому ключі. Американський і Російський стандарти шифрування DES і ДСТ28.147-89 - усі ці алгоритми є представниками симетричних криптосистем.

Асиметричні криптосистеми (***системи відкритого шифрування з відкритим ключем, public key systems***). Зміст даних криптосистем полягає в тому, що для за шифрування і

розшифрування використовуються різні перетворення. Одне з них (за шифрування) є абсолютно відкритим для усіх. Інше ж (розшифрування) залишається секретним. Таким чином, кожної, хто хоче зашифрувати, користається відкритим перетворенням. Але розшифрувати і прочитати це зможе лише той, хто воладіє секретним перетворенням. В даний момент у багатьох асиметричних криптосистемах вид перетворення визначається ключем. Т.ч. в користувача є два ключі - секретний і відкритий. Відкритий ключ публікується в загальнодоступному місці, і кожний, хто захоче послати повідомлення цьому користувачу - зашифровує текст відкритим ключем. Розшифрувати зможе тільки користувач із секретним ключем. Таким чином, пропадає проблема передачі секретного ключа. Однак, незважаючи на усі свої переваги, ці криптосистеми досить трудомісткі. Стійкість асиметричних криптосистем базується на алгоритмічних труднощах вирішити за малий час задачу. Якщо зловмиснику удасться побудувати такий алгоритм, то дискредетована буде вся система і всі повідомлення. У цьому складається головна небезпека асиметричних криптосистем на відміну від симетричних. Приклади - системи RSA, система Рабина і т.д.

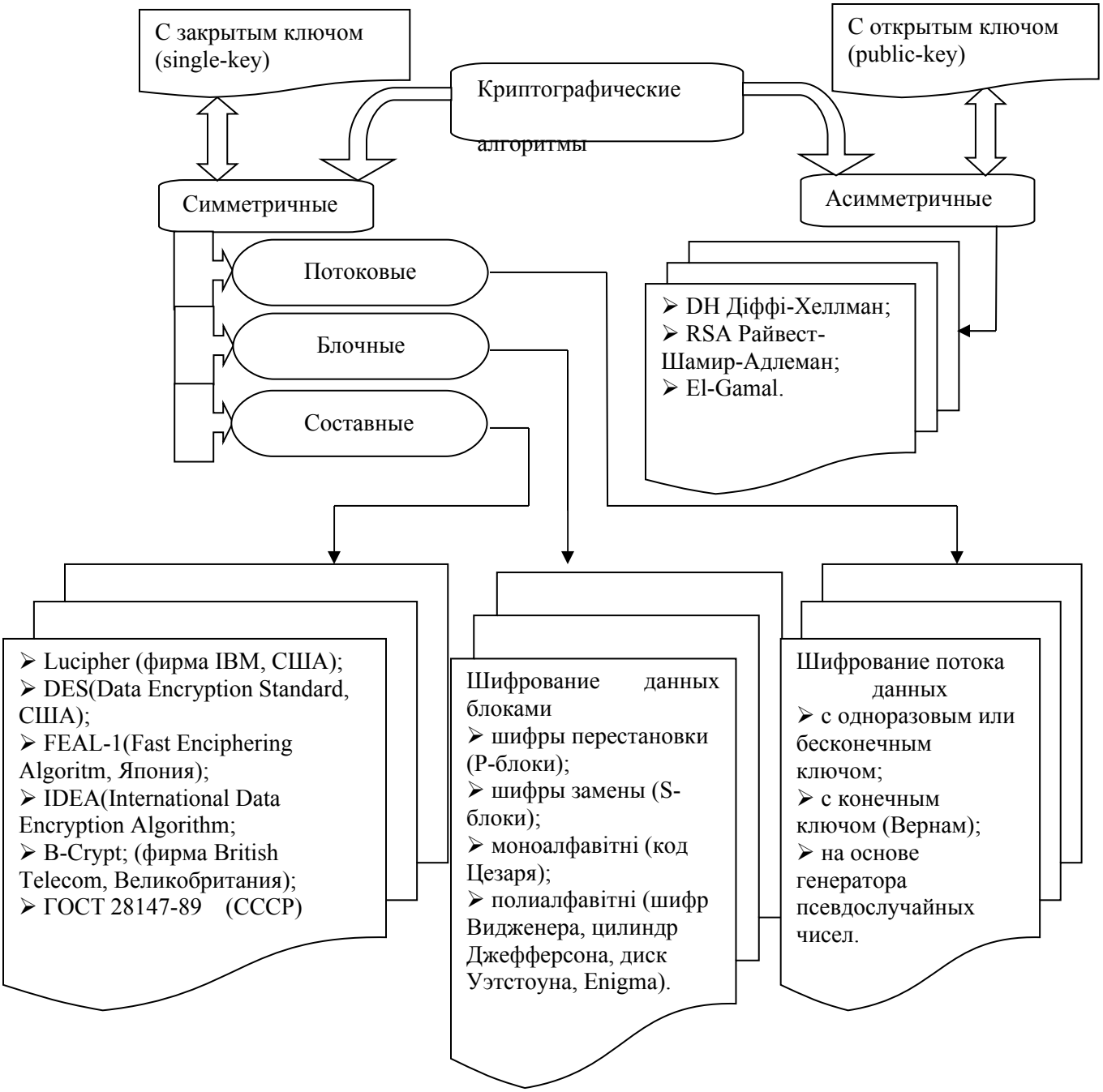


Рис1. Класифікація алгоритмів

Розглянемо докладно симетричні криптосистеми.

Постулатом для симетричних криптосистем є таємність ключа. Симетричні криптосхеми в даний час прийнято підрозділяти на **блокові** і **потокові**.

Блокові криптосистеми розбивають текст повідомлення (файлу, документа і т.д.) на окремі блоки і потім здійснюють перетворення цих блоків з використанням ключа.

Потокові криптосистеми працюють інакше. На основі ключа системи виробляється послідовність - так звана вихідна гама, що потім накладається на текст повідомлення. Таким чином, перетворення тексту здійснюється як би потоком в міру вироблення гами. Як правило, використовуються для військових, шифрування в засобах зв'язку і т.д.

Однак не слід вважати цей поділ закостенілим. Так, наприклад, при використанні деяких прийомів одержують із блокового шифру - потоковий і навпаки. А, наприклад, блоковий шифр із розміром вихідного блоку 8 біт (один символ) можна вважати потоковим. Загальну схему зв'язку у випадку симетричних криптосистем можна зобразити в такий спосіб:

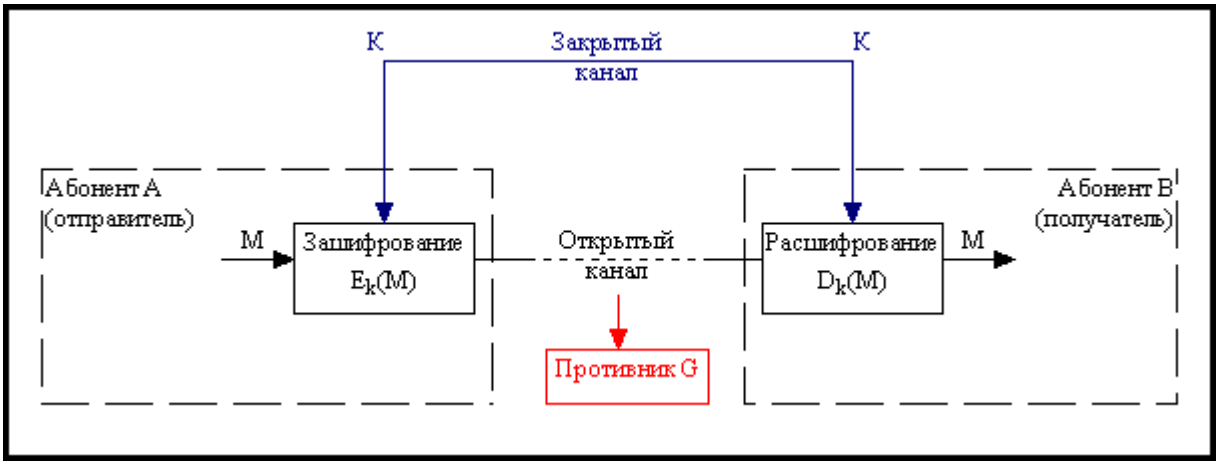


Рис. 2 – Схема криптосистеми.

На схемі М - відкритий текст, К - секретний ключ, переданий по закритому каналі, $E_к(M)$ - операція за шифрування, а $D_k(M)$ - операція розшифрування.

Виникає питання - чому не можна передавати дані відразу по закритому каналі, адже використовується ж він для передачі ключа? Відповідь проста. Як правило, закритий канал має дуже високу захищеність, але дуже низьку пропускну здатність, тобто працює на дуже низьких швидкостях. Набагато простіше передати по ньому маленький ключ, чим мегабайти інформації. Як закритий канал використовується кур'єрська доставка, обмін секретним ключем при зустрічі і т.д.

1 Блокові шифри

Блокові шифри оперують із блоками відкритого тексту. До них пред'являються наступні вимоги:

- достатня криптостійкість;
- простота процедур за шифрування і розшифрування;

- надійність.

Під криптостійкістю розуміють час, необхідний для розкриття шифру при використанні найкращого методу криптоаналіза. Надійність - частка інформації, дешифрована за допомогою криптоаналітичного алгоритму.

Саме перетворення шифру повинне використовувати наступні принципи (по К. Шеннону):

- ***Розсіювання (diffusion)*** - т.ч. зміна будь-якого знака відкритого тексту чи ключа впливає на велике число знаків шифротексту, що ховає статистичні властивості відкритого тексту;
- ***Перемішування (confusion)*** - використання перетворень, що утрудняють одержання статистичних залежностей між шифротектстом і відкритим текстом.

Практично всі сучасні блокові шифри є *композиційними* - складаються з композиції простих згідно з формулою

$$\mathbf{F=F_1 o F_2 o F_3 o F_4 o .. o F_n},$$

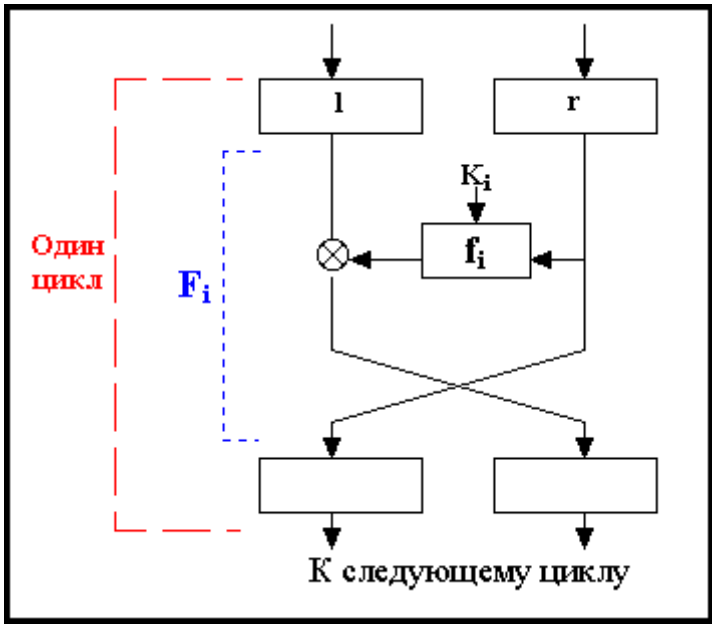
де **F-перетворення** шифру, **Fi-просте** перетворення, називане також і-им *циклом шифрування*.

Саме по собі перетворення може і не забезпечувати потрібних властивостей, але їхній ланцюжок дозволяє одержати необхідний результат. Наприклад, стандарт DES складається з 16 циклів. В іноземній літературі такі шифри часто називають пошаровими (layered). Якщо ж використовується те саме перетворення, тобто $\mathbf{F_i}$ постійно для $\forall$i, то такий композиційний шифр називають *ітераційним* шифром.

Найбільшу популярність мають шифри, улаштовані за принципом "шифру Фейстеля" (мережі Файстеля), яким чином:

1. Вхідний блок для кожного перетворення розбивається на дві половин: **p=(l,r)**, де **l-ліва**, **r-права**;
2. Використовується перетворення виду $\mathbf{F_i (l, r)=(r, l \otimes f_i (r))}$, де $\mathbf{f_i}$ - залежна від ключа $\mathbf{K_i}$ функція, а $\otimes$ - операція XOR чи інша.

Функція $\mathbf{f_i}$ називається *цикловою функцією*, а ключ $\mathbf{K_i}$, використовуваний для одержання функції $\mathbf{f_i}$ називається *цикловим ключем*. Як можна помітити, з цикловою функцією складається тільки ліва половина, а права залишається незмінною. Потім обидві половини міняються місцями. Це перетворення виконується кілька разів (кілька циклів). Виходом шифру є пара **(l,r)** Графічно усі виглядає в такий спосіб:



Як функція $\mathbf{f_i}$ виступає деяка комбінація перестановок, підстановок, зрушень, додавань ключа й інших перетворень. Так, при використанні підстановок інформація проходить через спеціальні блоки, називані *S-блоками (S-боксами, S-boxes)*, у яких значення групи бітов заміняється на інше значення. По такому принципі (з невеликими відмінностями) побудовані багато алгоритмів: DES, FEAL, серія LOKI.

В інших алгоритмах використовуються інші принципи. Так, наприклад, алгоритми, побудовані по *SP-принципі* (*SP-мережі*) здійснюють перетворення, пропускаючи блок через послідовність підстановок (*Substitutions*) і перстановок (*Permutations*). Звідси і назва - *SP-мережі*, тобто мережі "підстановок - перестановок". Прикладом такого алгоритму є Rijndael. Але всі перераховані алгоритми є композиційними. Саму ідею побудови криптографічно стійкою системи шляхом послідовного застосування простих криптографічних перетворень була висловлена Шенноном.
Розміри блоків у кожнім алгоритмі свої. DES використовує блоки по 64 біта (дві половинки по 32 біта), LOKI97 - 128 біт. При розмірі вихідних блоків до 8 біт шифр можна вважати потоковим.

Ключ має фіксовану довжину. Однак при прокручуванні хоча б 8 циклів шифрування з розміром блоку 128 біт навіть при простому додатку за допомогою XOR буде потрібно 8*128=1024 біта ключа. Не можна додавати в кожнім циклі те саме значення - це послабляє шифр. Тому для одержання послідовності ключових біт створюють спеціальний *алгоритм вироблення циклових ключів*. У результаті роботи цього алгоритму з вхідних біт ключа шифрування виходить масив біт визначеної довжини, з якого за визначеними правилами складаються циклові ключі. Кожен шифр має свій алгоритм вироблення циклових ключів.

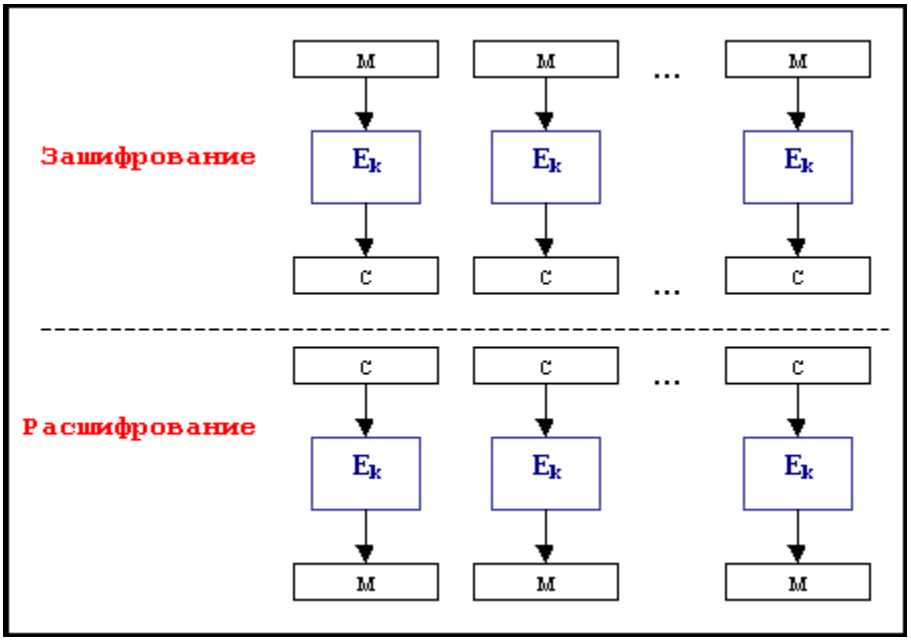
Режими роботи блокових шифрів.

Щоб використовувати алгоритми блокового шифрування для різних криптографічних задач існує кілька режимів їхньої роботи. Найбільш часто зустрічаються в практиці наступні режими:

- електронна кодова книга - ECB (Electronic Code Book);
- зчеплення блоків шифротексту - CBC (Cipher Block Chaining);
- зворотний зв'язок по шифротектсту - CFB (Cipher Feed Back);
- зворотний зв'язок по виходу - OFB (Output Feed Back);

Перед розглядом позначимо застосування шифру до блоку відкритого тексту як $E_k(M)=C$, де **k -** ключ, **M** - блок відкритого тексту, а **C** - шифротекст.

В режимі «***Електронна Кодова Книга (ECB)»*** вихідний текст розбивається на блоки, рівні розміру блоку шифру. Потім кожен з блоків шифрують незалежно від інших з використанням одного ключа шифрування. Графічно це виглядає так:



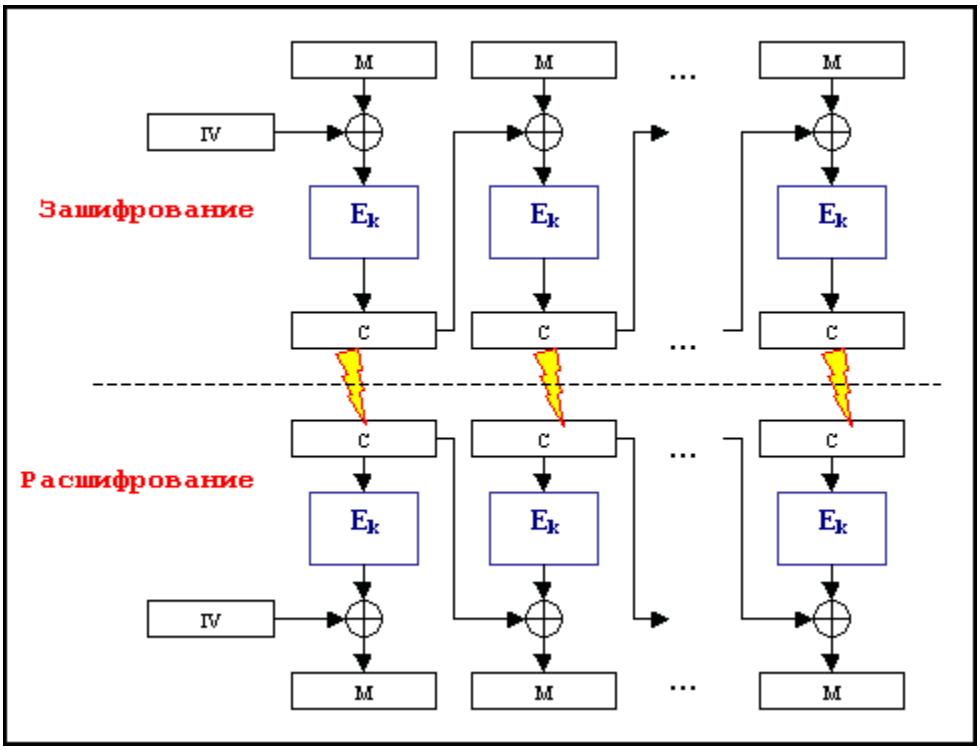
Безпосередньо цей режим застосовується для шифрування невеликих обсягів інформації, розміром не більш одного блоку чи для шифрування ключів. Це зв'язано з тим, що однакові блоки відкритого тексту перетворяться в однакові блоки шифротексту, що може дати зломщику (криптоаналітику) інформацію про зміст повідомлення. До того ж, якщо він припускає наявність визначених слів у повідомленні (наприклад, слово "До побачення"), те виходить, що він володіє як фрагментом відкритого тексту, так і відповідного шифротексту, що може сильно полегшити задачу взлому ключа. Основним достоїнством цього режиму є простота реалізації.

Режим ***«Зчеплення блоків шифротексту (CBC)»*** один з найбільше часто застосовних режимів шифрування для обробки великих кількостей інформації. Вихідний текст розбивається на блоки, а потім обробляється за наступною схемою:

1. Перший блок складається побітно по модулі 2 (XOR) з значенням IV - початковим вектором (Init Vector), що вибирається незалежно перед початком шифрування.

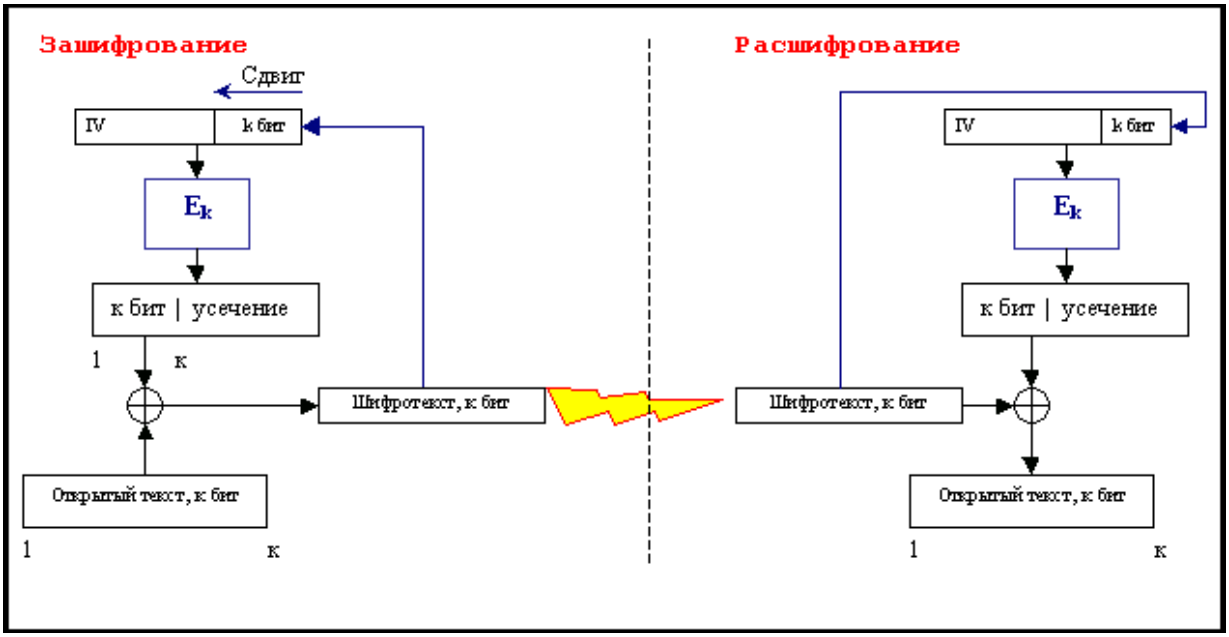
2. Отримане значення шифрується.
3. Отриманий у результаті блок шифротексту відправляється одержувачу й одночасно служить початковим вектором IV для наступного блоку відкритого тексту.

Расшифрування здійснюється в зворотному порядку. Графічно схема виглядає в такий спосіб:



Формулу, перетворення в режимі CBC можна представити як $\mathbf{C_i=E_k(M_i \oplus C_{i-1})}$, де **i** - номер відповідного блоку. Через використання такого зчеплення блоків шифротексту з відкритим текстом пропадають зазначені вище недоліки режиму ECB, оскільки кожен наступний блок залежить від усіх попередніх. Якщо під час передачі один із блоків шифротексту передасться з помилкою, то одержувач зможе коректно розшифрувати наступні блоки повідомлення. Проблеми виникнуть тільки з цим "бракованим" і наступним блоками. Однією з важливих властивостей цього режиму є "поширення помилки" - зміна блоку відкритого тексту змінює всі наступні блоки шифротексту. Оскільки останній блок шифротексту залежить від усіх блоків відкритого тексту, то його можна використовувати для контролю цілісності й автентичності (перевірки дійсності) повідомлення. Його називають *кодом аутентифікации повідомлення (MAC - Message Authentication Code)*. Він може захистити як від випадкових, так і навмисних змін у повідомленнях.

Режим «***Зворотний зв'язок по шифротексту (CFB)***» може використовуватися для одержання потокового шифру з блокового. Розмір блоку в даному режимі менше або дорівнює розміру блоку шифру. Схема даного режиму має вігляд:



Алгоритм роботи:

1. IV являє собою зсуновий регістр. Спочатку IV заповнюється деяким значенням, що називається сінхропосилкою, не є секретним і передається перед сеансом зв'язку одержувачу.
2. Значення IV шифрується.
3. Беруться перші k біт зашифрованого значення IV і складаються (XOR) з k бітами відкритого тексту ⇒ получаємо блок шифротексту з k біт.
4. Значення IV зрушується на k бітов уліво, а замість нього стає значення шифру.
5. Потім знову 2 пункт і т.д. до кінця.

Расшифрування аналогічно. Особливістю даного режиму є поширення помилки на весь наступний текст. Рекомендовані значення k: 1 =< k =< 8. Застосовується як правило для шифрування потоків інформації типу оцифрованої мови, відео.

Режим «***Зворотний зв'язок по виходу (OFB)***» примітний тим, що дозволяє одержувати потоковий шифр у його класичному виді. Принцип роботи схожий із принципом роботи режиму CFB, але зсувний регістр IV заповнюється не бітами шифротексту, а бітами, що виходять з під усікання.

Аутентифікація повідомлень за допомогою блокових шифрів.

Аутентифікація (authentication) - перевірка дійсності чого(чи кого). Може бути аутентифікація користувача, повідомлення і т.д. Необхідно відрізняти її від наступного поняття. ***Ідентифікація (identification)*** - деяке описове представлення якогось субьекту. Так, якщо хтось заявляє, що він - Вася Іванов, то він ідентифікує себе як "Вася Іванов". Але перевірити, чи у так це насправді (провести аутентифікацію) ми можемо тільки за

допомогою його паспорта. Отже, перевірити дійсність повідомлення за допомогою блокового шифру досить просто.

1. Відправник А хоче відправити деяке повідомлення $(a_1,..,a_t)$. Він зашифровує його на секретному ключі, що знає тільки він і одержувач, у режимі CBC чи CFB. А потім із шифротексту бере останній блок b_t з *до* біт.
2. Відправник А посилає повідомлення $(a_1, .. , a_t , b_t)$ одержувачу у відкритому виді чи зашифрувавши його на іншому ключі.
3. Одержувач В, одержавши повідомлення, $(a_1, .. , a_t , b_t)$, зашифровує $(a_1,..,a_t)$ у тім же режимі, що й А (повинна бути домовленість) на тім же секретному ключі (який знає тільки він і А).
4. Порівнюючи отриманий результат з b_t він засвідчується, що повідомлення відправив А, що воно не було підроблено на вузлі зв'язку (у випадку передачі у відкритому виді).

У даній схемі b_t є *кодом аутентифікации повідомлення (MAC)*. Для російського стандарту шифрування процес одержання коду аутентифікации називається роботою в режимі і*мітовставки*.

Режим шифрування повинний бути обов'язково з поширенням помилок (т.ч. CBC чи CFB). Необхідно використовувати шифр із достатньою довжиною блоку, а те може з'явитися ситуація, коли через невелике число використовуваних біт для аутентифікации можливо підмінити вихідне повідомлення і при знанні ключа одержати той же самий результат. Також очевидно, що схема спирається на те, що обоє абонента мають той самий секретний ключ, що одержали заздалегідь.

2 Потокові шифри

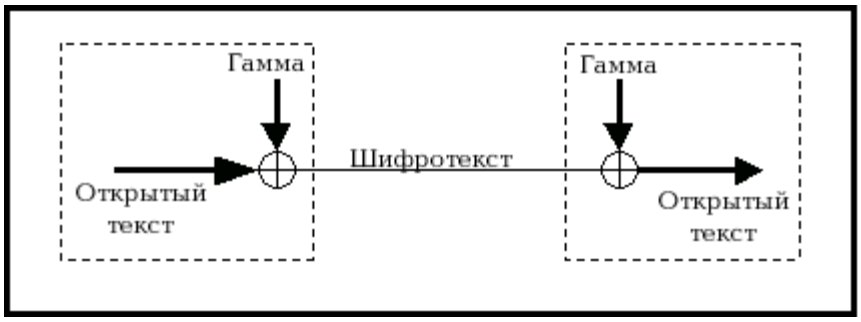
Шифрування в потокових шифрах здійснюється на основі додавання деякої *ключової послідовності* (*гами*) до відкритого тексту повідомлення. Додавання здійснюється по знаково за допомогою XOR. Рівняння за шифрування виглядає в такий спосіб:

$$c_i = m_i \oplus k_i \text{ для } i=1,2,3...$$

де c_i - знак шифротексту, m_i - знак відкритого тексту, k_i - знак ключової послідовності. Расшифрування виглядає так:

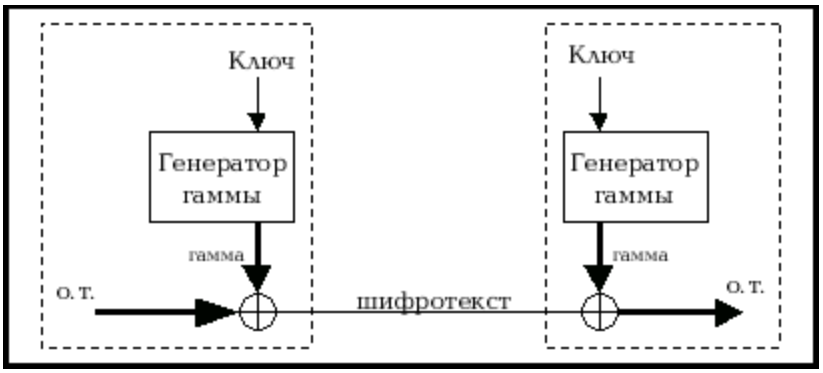
$$m_i = c_i \oplus k_i \text{ для } i=1,2,3...$$

Як знаки можуть виступати окремі біти, так і символи (байти). Таким чином, потокові шифри підходять для шифрування безупинних потоків даних - голосу, відео і т.д. У загальному виді схему шифру можна зобразити в такий спосіб:



Можна сказати, що шифрування здійснюється накладенням *гами*(шифрування *гаммуванням*). А сама гама є ключем шифрування. Але мати ключ, рівний по розмірі шифруємим даним здається проблематичним. Тому потокові шифри і виробляють вихідну гаму на основі деякого секретного ключа невеликого розміру, а значить основною задачею потокових шифрів є вироблення деякої послідовності (*вихідної гами*) для шифрування. Т.ч. вихідна гама є ключовим потоком для повідомлення. Потокові шифри класифікують у такий спосіб: *синхронні*, *що само синхронізуються (асинхронні)*.

Синхронні потокові шифри - ключовий потік (вихідна гама) виходить незалежно від вхідного і шифрованого текстів. У даному випадку ілюстрація міняється в наступну:



Шифр виробляє гаму на основі секретного ключа, вона складається з відкритим текстом і результат посилається іншому абоненту. Блок, що виробляє гаму називається *генератором* гами чи *псевдо випадковим генератором (гамми) - PRG(Pseudo Random Generator)*. Блоковий шифр у режимі OFB являє собою синхронний потоковий шифр.

Очевидно, що на вироблювану послідовність накладаються вимоги. Адже якщо в ній є великі послідовності нулів, то виходить, що в лінію при передачі передається текст повідомлення з відкритому виді. Якщо послідовність з одиниць - той же ефект (тому що ніщо не заважає супротивнику спробувати "простягнути" одиниці як гаму для перехопленого повідомлення. Результат - шматок відкритого тексту в супротивника). Тому найкращою гамою є випадкова послідовність. Однак не можна незалежно друг від друга згенерувати 2 однакові випадкові послідовності.

Генератори гами виробляють так називані *псевдовипадкові послідовності*, що залежать від ключа шифрування. Задача ставиться таким чином, щоб при наявності визначеної кількості бітов послідовності не можна було пророчити наступні біти. Крім

цього 1 і 0 на вході повинні бути рівноімовірні. Для досягнення цього послідовності досліджуються статистичними тестами.

Узагалі, генерування непередбачених псевдо імовірних послідовностей є однієї з важливих криптографічних задач на сьогоднішній день. Придумано безліч генераторів, статистичних тестів. Синхронні потокові шифри мають наступні властивості:

- *вимоги по синхронізації*. При використанні синхронних потокових шифрів одержувач і відправник повинні бути *синхронізовані* - тобто виробляти однакові значення ключового потоку для відповідних знаків переданого потоку даних. Якщо синхронізація порушиться (наприклад, унаслідок утрати знака при передачі), процес розшифрування не дасть коректного результату.
- *відсутність розмноження помилок*. Зміна знака шифртексту при передачі не викликає помилок при расшифруванні інших знаків шифротексту.
- *властивість активної атаки*. Як наслідок першої властивості, будь-яка вставка / видалення символу в шифротекст активним супротивником приводить до порушення синхронізації і виявляється одержувачем, що розшифровує повідомлення. Як наслідок другої властивості, активний супротивник може змінювати символи шифртексту і ці зміни приведуть до відповідних змін у відкритому тексті, одержуваному при расшифруванні. Тому необхідні додаткові механізми, що дозволяють запобігти це.

Розглянемо одну з ***атак*** на такі шифри. Нехай $O_1O_2O_3...$ - знаки відкритого тексту, $K_1K_2K_3...$ - знаки ключової послідовності, $C_1C_2C_3...$ - знаки шифротексту, отримані в такий спосіб:

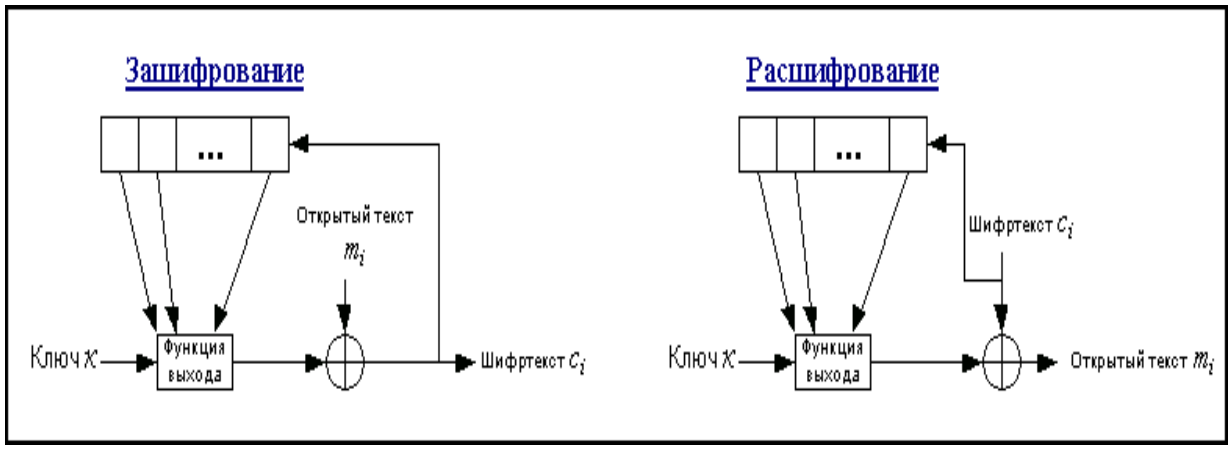
$$\begin{array}{ll} & O_1\ O_2\ O_3\ O_4\ \ldots \\ \oplus & \underline{K_1\ K_2\ K_3\ K_4\ \ldots} \\ & C_1\ C_2\ C_3\ C_4\ \ldots \end{array}$$

Припустимо, що при повторному шифруванні на тім же ключі відбулася вставка одного знака O':

$$\begin{array}{ll} & O_1\ O'\ O_2\ O_3\ \ldots \\ \oplus & \underline{K_1\ K_2\ K_3\ K_4\ \ldots} \\ & C_1\ C'_2\ C'_3\ C'_4\ \ldots \end{array}$$

Криптоаналітик супротивника перехопив обидві послідовності $C_1C_2C_3C_4$ і $C_1C'_2C'_3C'_4$. Склавши потім рівняння: $K_2 = C'_2 \oplus O'$; $O_2 = C_2 \oplus K_2$; $K_3 = C'_3 \oplus O_2$; $O_3 = C_3 \oplus K_3$ і підібравши значення одного знака O' він зможе прочитати повідомлення після цього знака. Оскільки в результаті досліджень у нього буде фрагмент ключової послідовності (гами), те він може спробувати відновити всю гаму й одержати повідомлення цілком. Звідси можна зробити висновок, що ***не можна двічі використовувати той самий ключ***.

Хоча опис атаки носить гіпотетичний характер, проте вона дуже і дуже реальна.
Потокові шифри, що само синхронізуються - кожен знак ключового потоку визначається фіксованим числом попередніх знаків шифротексту. Схематично це можна зобразити так:



Даному типу шифрів відповідають блокові шифри, що працюють у режимі CFB.

Лекція 13. Алгоритми шифрування з відкритим ключем.

Основною проблемою, з яким стикалася класична криптографія, була проблема *передачі ключа*. Оскільки сучасна криптографія вважає, що ключ потрібно постійно змінювати, те для цього варто мати надійний канал зв'язку для передачі ключа. Це добре, якщо передачу ключа потрібно здійснити в межах одного міста. Ну а якщо ключ треба передати в іншу країну, тим більше, що передачу ключа потрібно проводити таки часто? Відсутністю цих труднощів характеризується концепція *відкритого ключа*, розвита в 1976 році американськими математиками *Діффі*, *Хелманом* і *Мерклем*.

В основі пристрою асиметричних криптосистем лежить поняття *однобічної функції*.

Однобічна (односпрямована) функція - це функція *f, яка* здійснює відображення **X➔Y**, де **X** і **Y** - довільні безлічі, і задовольняє наступним умовам:

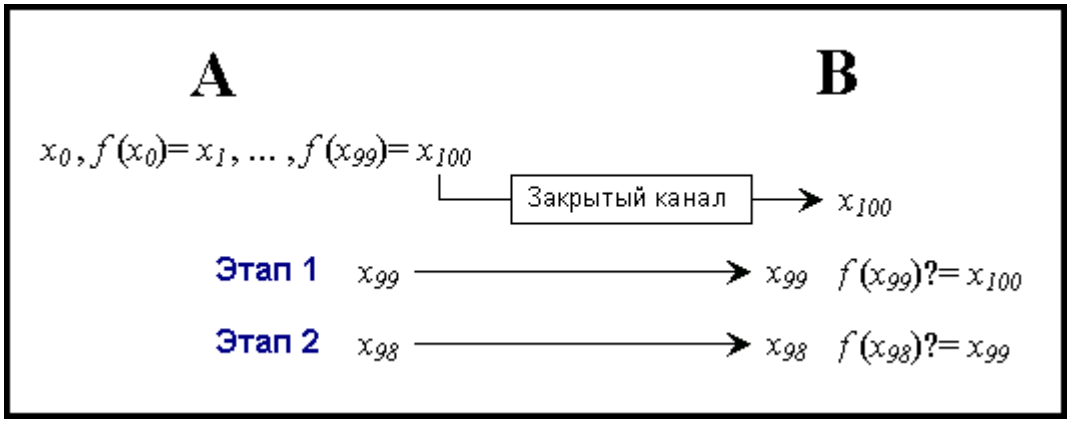
1. . $x \in$ **X** (області визначення) і легко обчислити $y=f(x)$, у$\in$**Y**.
2. Майже для будь-якого у$\in$**Y** (області значення) обчислювально неможливо знайти $f^{-1}(y)$ (тобто x, для якого $y=f(x)$).

Чому "майже для будь-якого"? Тому, що якщо взяти деякий х і обчислити для нього $y=f(x)$, то ми ж вже будемо знати, що отриманому у відповідає узятий нами х. Збережемо ці 2 значення і якщо коли-небудь ми зштовхнемося з таким у, то ми спокійно знайдемо х.

Прикладом однобічної функції може служити обчислення $a^x \bmod n$, де a і n - деякі числа. Така задача називається NP-задачею дискретного логарифмування. В даний час немає ефективних алгоритмів, що вирішують цю задачу для великих чисел. Узагалі,

приведений приклад можна назвати однобічною функцією сьогодні, оскільки якщо з'явиться такий алгоритм чи раптом сильно збільшаться обчислювальні потужності, те така задача стає розв'язуваною. Тому пошук дійсно однобічних функцій чи навіть доказ їхнього існування є однієї з важливих задач криптографії.

Прикладом застосування однобічної функції може служити наступна схема ідентифкації. Абонент А виробляє наступну послідовність: $x_0, f(x_0)=x_1, \ldots, f(x_{99})=x_{100}$. Потім x_{100} передається по секретному каналі (чи при зустрічі) абоненту В. Коли А необхідно ідентифікувати себе, він передає по відкритому каналуВ x_{99} . В перевіряє: $f(x_{99})=?x_{100}$. Наступного разу А передасть x_{99} і В перевірить $f(x_{98})=?x_{99}$ і т.д. Перехоплення повідомлень на *і-ом* етапі у відкритому каналі нічого не дасть зловмиснику, тому що він не зможе одержати відповідне значення x_{i-1}, щоб наступного разу ідентифікувати себе як абонента А. Дана схема представлена на наступному малюнку:



На основі однобічних функцій із секретом і будуються асиметричні криптосистемы. Так, алгоритм зашифрування з відкритим ключем можна розглядати як однобічну функцію із секретом, а секретом для цієї функції є секретний ключ, використовуючи який можна розшифрувати повідомлення.

Криптосистема Рабина

Опис цієї системи почнемо з генерування ключів. Отже, потрібно вибрати два різних великих простих числа *p* і *q*. Великими будемо вважати числа розрядністю не менш 200 біт. Також нагадаємо, що простими називаються числа, що не мають дільників, крім одиниці і самого себе. Отже, далі розраховуємо добуток: $n=p*q$. Відкритим ключем будемо вважати число n, а таємним ключем будуть числа p і q.

Процес шифрування здійснюється блоками. Для цього вхідний текст потрібно записати в цифровій формі і розбити на блоки так, щоб кожен блок не перевищував n. Цифрова форма тексту буде представляти кожен символ як число, що не перевищує n. Наприклад, кожній букві алфавіту можна зіставити її порядковий номер.

Далі до кожного блоку застосовуємо процедуру

$$Ci=E(Mi)=Mi \bmod n.$$

Блоки *Ci* будуть утворювати криптотекст.

Тепер розглянемо процес дешифрування. Якщо *E(M)=3*, то блок M буде коренем квадратним числа 3 по модулі n. Нагадаємо, що коренем квадратним числа x по модулі n називається число y, для якого виконується співвідношення: *y = (x^2) mod n*. За умови, що числа C и n будуть взаємнопрості, ми одержимо рівно чотири таких корені.

Існують алгоритми знаходження цих коренів, але для їхнього успішного застосування треба знати таємний ключ, тобто числа p и q. Саме процедура знаходження квадратного кореня і використовується при дешифруванні по системі Рабина. Після дешифрування з чотирьох отриманих коренів вибирається один, котрому буде відповідати осмислена текстова інформація.

Усі добре, але що робити, якщо потрібно передати цифрову інформацію? У цьому випадку разом із криптотекстом треба передавати додаткову незашифровану інформацію. Крім цього слід зазначити, що якщо числа C и n не будуть взаємопрості, то конкуруюча організація може одержати таємний ключ, тобто факторизировать число n. Тому шифруемый текст такого типу потрібно виключити.

Надійність криптосистемы Рабина полягає в тому, що задача знаходження квадратного кореня по модулі $n = p*q$ не менш важка, чим факторизация числа $n = p*q$.

Приклад 1.

Для початку згенеруємо відкритий і таємний ключі. Виберемо $p = 7$, $q = 11$, тоді $n = p*q = 77$. Отже, $n = 77$ буде відкритим ключем, а $p = 7$ і $q = 11$ — таємним.

Нехай нам потрібно зашифрувати слово «Так». Представимо його в цифровому виді, привласнивши кожній букві її порядковий номер в алфавіті.

Тоді одержимо, що $M = 51$.

Зашифровуємо: $3 = (51^2) \bmod 77 = 2601 \bmod 77 = 60$.

А тепер розшифровуємо. Для цього знайдемо квадратні корені числа 60 по модулі 77. Це будуть числа 26, 37, 40, 51. Як бачимо, лише останньому числу буде відповідати нормальна текстова інформація (слово «ДА»).

Алгоритм RSA

Одержувач повідомлення виконує генерацію відкритого ключа (пари чисел N,e) і закритого ключа (d) у такий спосіб:

1) Беремо два простих числа p і q p=11; q=3;
2) Визначаємо першу частину відкритого ключа N = p*q = 3*11 = 33;
3) Визначаємо другу частину відкритого ключа Z=(p-1)*(q-1)=10*2=20;
4) Знайдемо число e взаимнопростое з Z (e=7. Числа називаються взаємно простими, якщо їх найбільший загальний дільник дорівнює 1.
5) Знайдемо число d, таке що воно задовольняє умові (d*e) mod z =1, d=3, наприклад, використовуючи алгоритм Эвклида;

Після цього ключ повідомляється бажаним відправникам сообщаний.

6) За умовою алгоритму можна кодувати числа по модулі не переважаючі N. У даному випадку <= 33. Тому для кодування будемо використовувати російський алфавіт. Для простоти візьмемо порядкові номери букв і закодуємо слово «ОВЕН»

   О=16; В=3; Е=6; Н=15;

7) Будемо в ролі відправника шифрувати повідомлення по формулі Ci=Si^e mod N;

C1=16^7 mod 33=25;
C2=3^7 mod 33=9;
C3=6^7 mod 33=30;
C4=15^7 mod 33=27;

8) Передаємо зашифрованый текст.
9) Для дешифрації як одержувач будемо використовувати формулу Mi= Ci^d mod N;

M1=25^3 mod 33=16;
M2=9^3 mod 33=3;
M3=30^3 mod 33=6;
M4=27^3 mod 33=15;

У результаті ми одержали відповідність з вихідним текстом.

Алгоритм Эль-Гамаля

Основна ідея ElGamal полягає в тому, що не існує ефективних методів рішення порівняння $a^x == b \pmod p$

Ключі будуються наступним образом:

1. Вибирається деяке число P;
2. Вибирається число W менше P-1, так, щоб $W^{P-1} = 1 \pmod P$
3. Вибирається 1<=X<=P-1;
4. $Y := W^X \bmod P$;
5. Секретний ключ -- (X,P);
6. Відкритий ключ -- (Y,W,P).

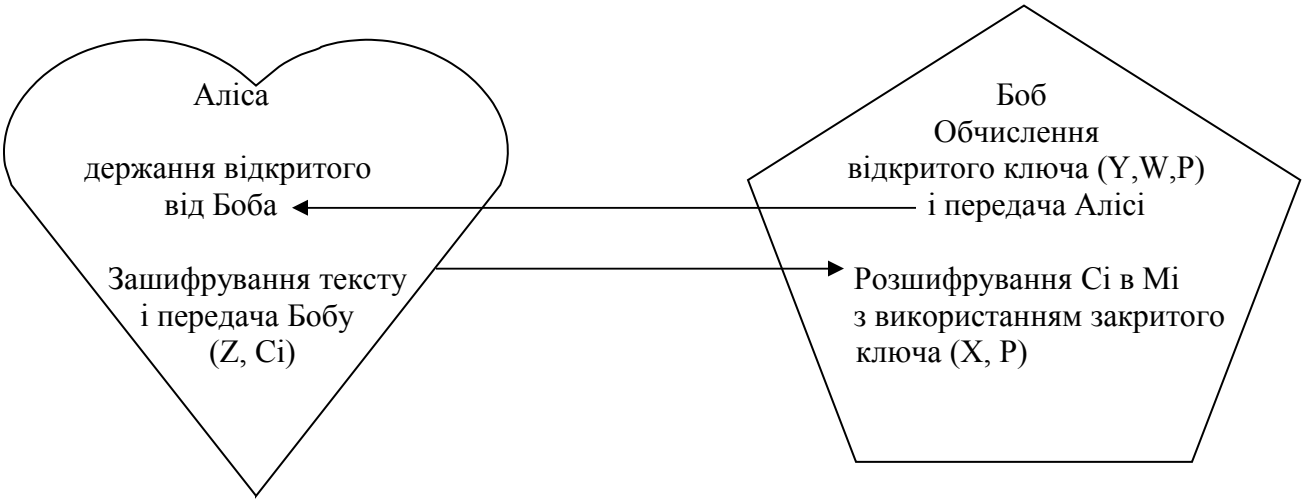
Шифрування блоку B протікає в такий спосіб (відкритий ключ):

1. Береться випадкове число 1<=K<=P-1;
2. $R := Y^K \bmod P$;
3. $Z := W^K \bmod P$;
4. C := B*R;
5. Результат -- пара (C,Z).

Расшифрование протікає в такий спосіб (секретний ключ):

1. $R := Z^X \bmod P$;
2. B := C/R.

Загальна схема алгоритму виглядає наступним чином



Вхідними даними для кодування є відкритий текст і ключ, а для декодування – зашифрований текст і ключ. Для приклада розглянемо відкритий текст, розміром у 12 байт, наприклад “Енциклопедія”. Даний відкритий текст необхідно перевести в послідовність кодів ASCII (показано при шифруванні).

1) Вибираємо P=65477 (просте число в інтервалі 256<>65536. Нижній інтервал такий, тому що він обмежує розмір числа яке можна кодувати, а верхній інтервал щоб числа були розміру WORD).
2) Вибираємо число W=60422, таке щоб $W^{p-1}=1(\text{mod } P)$;
3) Знаходимо закритий ключ X=34567(довільне число в проміжку 1<==>p-1)
4) Знаходимо $Y=W^X \text{ mod } P$; При прямій реалізації даної формули виникають складності з представленням інформації і наступних проміжних результатів. Доцільно виконати зведення в ступінь послідовно:

(((W mod P)*W)mod P)*W)mod P)...…….....

Одержуємо Y=(60422^32567)mod 65447=15866.

Шифрування інформації:

1) Знаходимо ДО=54321(довільне число в проміжку 1<==>p-1)
2) Знаходимо $R=Y^K \text{ mod } P$= (15866^54321)mod 65447=34158, використовуючи послідовність п.4;
3) Знаходимо $Z=W^k \text{ mod } P$= (60422^54321)mod 65477= 55249;
4) Кодуємо відкритий текст Mi відповідно до формули:

$$Ci = Mi*R;$$

C1=133*34158=4543014;(Е)
C2=141*34158=4816278;(Н)
C3=150*34158=5123700;(Ц)
C4=136*34158=4645488;(И)
C5=138*34158=4713804;(ДО)
C6=139*34158=4747962;(Л)
З7=142*34158=4850436; (ПРО)
C8=143*34158=4884594; (П)
C9=133*34158=4543014; (Е)

C10=132*34158=4508856; (Д)
C11=73*34158=2493534; (І)
C12=159*34158=5431122;(Я)
Дешифрування:
Знаходимо R = Z^x mod P = (55249^34567)mod 65447=34158;
Декодируем по формулі Mi = Ci / R;
M1=4543014/34158=133;
M2=4816278/34158=141;
M3=5123700/34158=150;
M4=4645488/34158=136;
M5=4713804/34158=138;
M6=4747962/34158=139;
M7=4850436/34158=142;
M8=4884594/34158=143;
M9=4543014/34158=133;
M10=4508856/34158=132;
M11=2493534/34158=73;
M12=5431122/34158=159;
Результати, як бачимо, збігаються.

Задача дискретного логарифмування при правильному виборі цілих чисел настільки складна, що дозволяє сподіватися на практичну неможливість відновлення числа **x** (індивідуального ключа підписування) по числу **b** (застосовуваному як ключ перевірки).

Якщо зломник має в розпорядженні обчислювальну систему загальною потужністю 1 мільярд ($\mathbf{10^9}$) операцій у секунду, а це потужність сучасного суперкомп'ютера типу Cray-3, то:

- за добу безупинної роботи такої системи може бути вирішена задача складністю близько 100 трильйонів (чи $\mathbf{10^{14}}$) операцій,
- за місяць - близько $\mathbf{3.10^{15}}$,
- за рік - близько $\mathbf{3.10^{16}}$,
- за десять років - близько $\mathbf{3.10^{17}}$,
- за тридцять років - близько $\mathbf{10^{18}}$ операцій.

Таким чином, навіть якщо допустити, що потенційний зломщик має у своєму розпорядженні обчислювальну систему, еквівалентної по потужності тисячі ($\mathbf{10^3}$) суперкомп'ютерів типу Cray-3, на виконання обчислень обсягом $\mathbf{10^{21}}$ операцій йому треба було б не менш тридцяти років безупинної роботи системи. Сучасні процесори класу Pentium дозволяють за частки секунди обчислювати і перевіряти цифрові підписи з ключами довжиною до **512 байт**, а стійкість більшості широко застосовуваних методів цифрового підпису при такій довжині ключів свідомо перевершує всі розумні вимоги (більш ніж $\mathbf{10^{50}}$).

Оскільки криптосистема працює з дуже великими числами і виконує такі "важкі" дії, як зведення в ступінь, то швидкодія криптосистеми невисока. Асиметричні криптосистемы уступають у швидкості симетричним саме "завдяки" використанню таких перетворень. Існують різні методи підвищення швидкості RSA. Наприклад, як відкритий ключ *e* береться *e*=3. Тоді для того, щоб зашифрувати, необхідно всего 3

множення. Правда в цьому випадку виходить велике *d*, але припускають, що у розшифровника є час. Однак використання малої експоненти *e* приводить до проблем з маленькими повідомленнями, таких, що $M<n^{1/e}$, оскільки в цьому випадку *M* може бути отримане із шифртекста $C=M^e \bmod n$ шляхом обчислення кореня *e-ої* ступеня з *C*.

У тому випадку,якщо передане повідомлення дуже велике, воно розбивається на блоки, кожний з який потім представляється у виді числа і шифрується за допомогою RSA. Для того, щоб запобігти чи заміні перемішування блоків у повідомленні супротивником можливе використання режиму CBC блокових шифрів.

Одним з різновидів криптосистем з відкритим ключем є вірогідністне шифрування, розроблене *Шаф Гольвассером* і *Сильвио Минелли*. Його суть полягає в тому, щоб алгоритм шифрування E підкорити вірогідностним моделям. Для приклада, у системі RSA не «маскуються» 0 і 1. Отож, цю проблему успішно вирішують вірогідностні алгоритми, оскільки вони ставлять у відповідність відкритому тексту M не просто криптотекст C, а деякий елемент із безлічі криптотекстов *CM*. При цьому кожен елемент цієї безлічі вибирається з деякою імовірністю. Іншими словами, для будь-якого відкритого тексту M результат роботи алгоритму E буде випадковою величиною. Може показатися, що в цьому випадку дешифрувати інформацію буде неможливо.

Лекція 14. Введення в брандмауери.

Більшість корпоративних мереж захищені по периметру настроєними брандмауерами, що захищають внутрішніх користувачів від самих себе й від хакерів.

Брандмауер (firewall) представляє собою сукупність систем, що забезпечують рівень розмежування доступу, який досягається шляхом керування трафіком згідно до встановлених правил поведінки. Брандмауер пропускає тільки ту частину трафіка, що явно дозволена адміністратором і блокує всі інші.

На ринку домінують два типи брандмауерів:

- пакетні фільтри, або шлюзами фільтрації пакетів (packet filter gateway);
- програмні проксі (application proxy).

Прикладом першого з них є Firewall від компанії Check Point, а другого - Microsoft Proxy Server.

Пакетні фільтри повністю прозорі для користувачів і досить продуктивні, однак недостатньо надійні. Фактично, вони представляють собою різновид маршрутизатора, що приймає пакети як ззовні, так і зсередини мережі, і вирішає як з ними поступити - пропустити далі або знищити, при необхідності повідомивши відправника. Більшість брандмауерів цього типу працює на IP-рівні, причому повнота підтримки IP-протоколу і якість фільтрації залишають бажати кращого. На домашніх комп'ютерах такі брандмауери ще мають сенс, але при наявності навіть простого маршрутизатора вони лише роблять систему дорожчою, тому що ті ж самі правила фільтрації пакетів можна задати й на маршрутизаторі.

Програмні проксі являють собою звичайні проксі-сервера, що прослуховують задані порти (наприклад, 25, 110, 80) і підтримують взаємодію із заздалегідь обговореним переліком мережних сервісів. На відміну від фільтрів, що передають IP-пакети "як є", проксі самостійно збирають TCP-пакети, достають з них дані користувача, наклеюють на них новий заголовок, і знову розбирають отриманий пакет на IP, при необхідності здійснюючи трансляцію адрес. Якщо брандмауер не містить помилок, обдурити його на мережному рівні вже не вдасться. Він приховує від атакуючих структуру внутрішньої мережі - зовні залишається лише брандмауер. А для досягнення найвищої захищеності адміністратор може організувати на брандмауері додаткові процедури авторизації й аутентифікації супротивника заздалегідь. Це - переваги.

Але програмні проксі незручні, оскільки обмежують користувачів у виборі програм. Вони працюють набагато повільніше пакетних фільтрів і знижують продуктивність особливо на швидких каналах. Тому, головним чином ми будемо говорити про пакетні фільтри, залишивши програмні проксі осторонь.

Брандмауери обох типів звичайно містять у собі версію системи визначення вторгнень (Intruder Detection System або скорочено IDS), що аналізує характер мережних запитів і виявляє потенційно небезпечні дії:

- звертання до неіснуючих портів (характерно для сканування);
- пакети з TTL рівним одиниці (характерно для трасування) і т.д.

Все це ускладнює атаку й хакеру доводиться діяти дуже обережно, оскільки будь-який невірний крок відразу видасть його з головою. Однак інтелектуальність інтегрованих систем розпізнавання досить невелика й більшість адміністраторів перекладає це завдання на плечі спеціалізованих пакетів, таких наприклад, як Real Secure від Internet Security System.

Залежно від конфігурації мережі, брандмауер може бути встановлений як на виділений комп'ютер, так і ділити системні ресурси з ким-небудь ще.

Персональні брандмауери, розповсюджені у світі Windows, у більшості випадків установлюються на сам комп’ютер, що необхідно захистити. Якщо це пакетний фільтр, реалізований без помилок, то захищеність системи нітрохи не страждає й атакувати її, так само як і на виділеному брандмауері. Локальні програмні проксі захищають комп'ютер

лише від деяких типів атак (наприклад, блокують засилання троянів через IE), залишаючи систему повністю відкритою. В UNIX-подібних системах пакетний фільтри присутній споконвічно, а в штатний комплект поставки входить велика кількість різноманітних проксі - серверів, тому придбати додаткове програмне забезпечення не потрібно.

Від чого захищає брандмауер

Пакетні фільтри дозволяють:

- закривати всі вхідні/вихідні TCP-порти;
- повністю або частково блокувати деякі протоколи (наприклад, ICMP);
- перешкоджати установці з'єднань із даними IP-адресами й т.і.

Правильно сконфігурована мережа повинна складатися із двох зон:

1. внутрішньої корпоративної мережі (corporative network), обгородженої брандмауером і робочими станціями, мережними принтерами, intranet-серверами, серверами баз даних й інших ресурсів подібного типу;
2. демілітаризованої зони (demilitarized zone або, скорочено, DMZ), у якій розташовані публічні сервера, які повинні бути доступні з Інтернет.

Брандмауер повинен:

- Закривати всі порти, крім тих, що належать публічним мережним службам (HTTP, FTP, SMTP і т.д.);
- Пакети, що приходять на заданий порт, відправляти тим і тільки тим вузлам, на яких установлені відповідні служби (наприклад, WWW-сервер розташован на вузлі A, а FTP-сервер на вузлі B, те пакет, спрямований на 80 порт вузла B повинен блокуватися брандмауером);
- Блокувати вхідні з'єднання із зовнішньої мережі, спрямовані в корпоративну мережу (у цьому випадку користувачі мережі не зможуть працювати із зовнішніми FTP-серверами в активному режимі);
- Блокувати вихідні з'єднання з DMZ-зони, спрямовані у внутрішню мережу (крім FTP- і DNS-сервера, яким вихідні з'єднання необхідні);
- Блокувати вхідні з'єднання з DMZ-зони, спрямовані у внутрішню мережу (якщо цього не зробити, то атакуючий, що захопив керування одним з публічних серверів, безперешкодно проникне й у корпоративну мережу);
- Блокувати вхідні з'єднання в DMZ-зону із зовнішньої мережі по службових протоколах, що часто використається для атаки (наприклад, ICMP, щоправда, повне блокування ICMP створює більші проблеми, зокрема, перестає працювати ping і стає неможливим автоматичне визначення найбільш кращого MTU);
- Блокувати вхідні/вихідні з'єднання з портами й/або IP-адресами зовнішньої мережі, заданими адміністратором;

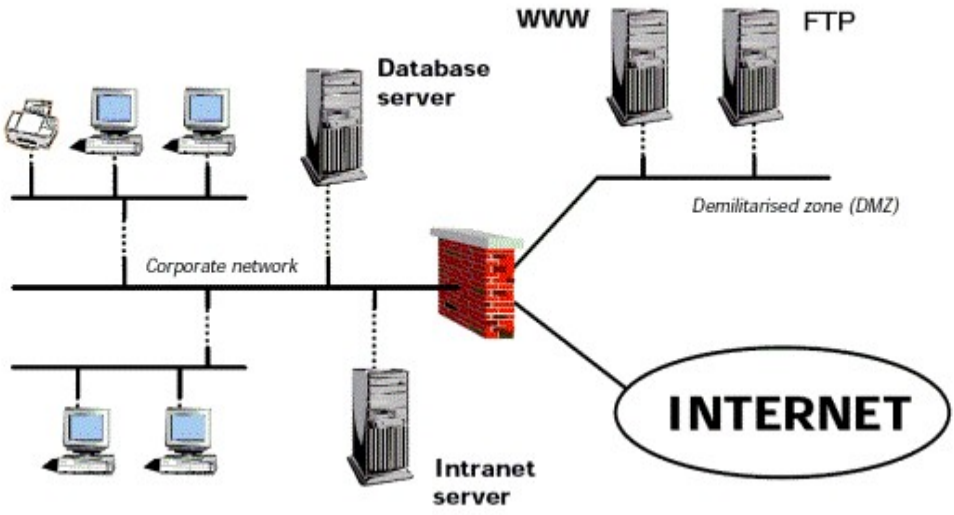


Рис. 1 – Типова структура локальної мережі.

Фактично роль брандмауера зводиться до огородження корпоративної мережі від усяких цікавих, блукаючих по просторах Інтернет. Якщо клієнт корпоративної мережі використає уразливу версію браузера або клієнта електронної пошти (а більшість програмних забезпечень уразливе!), що атакує досить заманити його на троянську WEB-сторінку або послати йому лист із вірусом усередині й через короткий час локальна мережа виявиться уражена. Навіть якщо вихідні з'єднання з корпоративної мережі заборонені, shell-код зможе скористатися вже встановленим TCP-з'єднанням, через яке він і був закачаний на атакований вузол, передаючи хакеру можливість дистанційного керування системою.

Брандмауер може й сам бути об'єктом атаки, адже він, як і всяка складна програма, не обходиться без дір. Діри в брандмауерах виявляються практично щороку.

Виявлення й ідентифікація брандмауера

Запорукою успішної атаки є своєчасне виявлення й ідентифікація брандмауера або в загальному випадку - системи виявлення вторгнень, але ми будемо виходити з того, що вона сполучена із брандмауером.

Більшість брандмауерів відкидають пакети TTL (Time To Live - час життя), блокуючи тим самим трасування маршруту, чим і викривають себе. Аналогічним образом надходять і деякі маршрутизатори, однак, як уже говорилося вище, між маршрутизатором і пакетним фільтром немає принципової різниці.

Відстеження маршруту звичайно здійснюється утилітою **`traceroute`**, що підтримує трасування через протоколи ICMP й UDP, причому ICMP блокується набагато частіше. Вибравши вузол, свідомо захищений брандмауером (наприклад, www.intel.ru), спробуємо відстежити до нього маршрут:

```
$traceroute -I wwww.intel.ru
```

Трасування маршруту до bouncer.glb.intel.com [198.175.98.50]
с максимальним числом стрибків 30:

```
1  1352 ms   150 ms   150 ms  62.183.0.180
2   140 ms   150 ms   140 ms  62.183.0.220
3   140 ms   140 ms   130 ms  217.106.16.52
4   200 ms   190 ms   191 ms  aksai-bbn0-po2-2.rt-comm.ru [217.106.7.25]
5   190 ms   211 ms   210 ms  msk-bbn0-po1-3.rt-comm.ru [217.106.7.93]
```

```
6   200 ms   190 ms   210 ms  spb-bbn0-po8-1.rt-comm.ru [217.106.6.230]
7   190 ms   180 ms   201 ms  stockholm-bgw0-po0-3-0-0.rt-comm.ru [217.106.7.30]
8   180 ms   191 ms   190 ms  POS4-0.GW7.STK3.ALTER.NET [146.188.68.149]
9   190 ms   191 ms   190 ms  146.188.5.33
10   190 ms   190 ms   200 ms  146.188.11.230
11   311 ms   310 ms   311 ms  146.188.5.197
12   291 ms   310 ms   301 ms  so-0-0-0.IL1.DCA6.ALTER.NET [146.188.13.33]
13   381 ms   370 ms   371 ms  152.63.1.137
14   371 ms   450 ms   451 ms  152.63.107.150
15   381 ms   451 ms   450 ms  152.63.107.105
16   370 ms   461 ms   451 ms  152.63.106.33
17   361 ms   380 ms   371 ms  157.130.180.186
18   370 ms   381 ms   441 ms  192.198.138.68
19     *        *        *     Перевищений інтервал очікування для запиту.
20     *        *        *     Перевищений інтервал очікування для запиту.
```

Лістинг 1 – Трасування маршруту, що зупиняється на брандмауері (маршрутизаторі)

Трасування доходить до вузла 192.198.138.68, а потім зупиняється, що вказує на наявність маршрутизатора або брандмауера. Виберемо для трасування інший вузол, наприклад, www.zenon.ru:

```
$traceroute -I wwww.zenon.ru.
Трасування маршруту до distributed.zenon.net [195.2.91.103]
с максимальним числом стрибків 30:

  1  2444 ms  1632 ms  1642 ms  62.183.0.180
  2  1923 ms  1632 ms  1823 ms  62.183.0.220
  3  1632 ms  1603 ms  1852 ms  217.106.16.52
  4  1693 ms  1532 ms  1302 ms  aksai-bbn0-po2-2.rt-comm.ru [217.106.7.25]
  5  1642 ms  1603 ms  1642 ms  217.106.7.93
  6  1562 ms  1853 ms  1762 ms  msk-bgw1-ge0-3-0-0.rt-comm.ru [217.106.7.194]
  7  1462 ms   411 ms   180 ms  mow-b1-pos1-2.telia.net [213.248.99.89]
  8   170 ms   180 ms   160 ms  mow-b2-geth2-0.telia.net [213.248.101.18]
  9   160 ms   160 ms   170 ms  213.248.78.178
 10   160 ms   151 ms   180 ms  62.113.112.67
 11   181 ms   160 ms   170 ms  css-rus2.zenon.net [195.2.91.103]
Трасування завершене.
```

Лістинг 2 – Успішне завершення трасування ще не є свідчення відсутності брандмауера

Цього разу трасування проходить нормально. Виходить, що ніякого брандмауера навколо zenon'а немає? Дуже може бути, але для впевненої відповіді нам потрібна додаткова інформація. Вузол 195.2.91.193 належить мережі класу З (три старших біти IP-адреси рівні 110) і, якщо ця мережа не захищена брандмауером, більшість її вузлів повинні відгукуватися на ping, що в цьому випадку й відбувається. Сканування виявляє 65 відкритих адрес. Отже, або маршрутизатора тут ні, або він безперешкодно пропускає наш ping.

При бажанні можна спробувати просканувати порти, однак, по-перше, наявність відкритих портів ще ні про що не говорить (може, брандмауер блокує лише один порт, наприклад, захищає дірявий RPC від зазіхань ззовні), а по-друге, при скануванні хакеру буде важко залишитися непоміченим. З іншого боку, порти сканують всі, кому не лінь й адміністратори вже давно не обертають на це уваги.

Утиліта **nmap** (популярний сканер портів) дозволяє виявляти деякі із брандмауерів, установлюючи статут порту в "firewalled". Таке відбувається щораз, коли у відповідь на SYN, вилучений вузол повертає ICMP пакет типу 3 з кодом 13 (Admin Prohibited Filter) з дійсною IP-адресою брандмауера в заголовку (nmap його не відображає). Якщо повернеться SYN/ACK - скануємий порт відритий. RST/ACK указує на закритий або заблокований брандмауером порт. Не всі брандмауери генерують RST/ACK при спробі

підключення до заблокованих портів (Check Point Firewall - генерує), деякі відсилають ICMP повідомлення, як було показано вище, або не посилають взагалі.

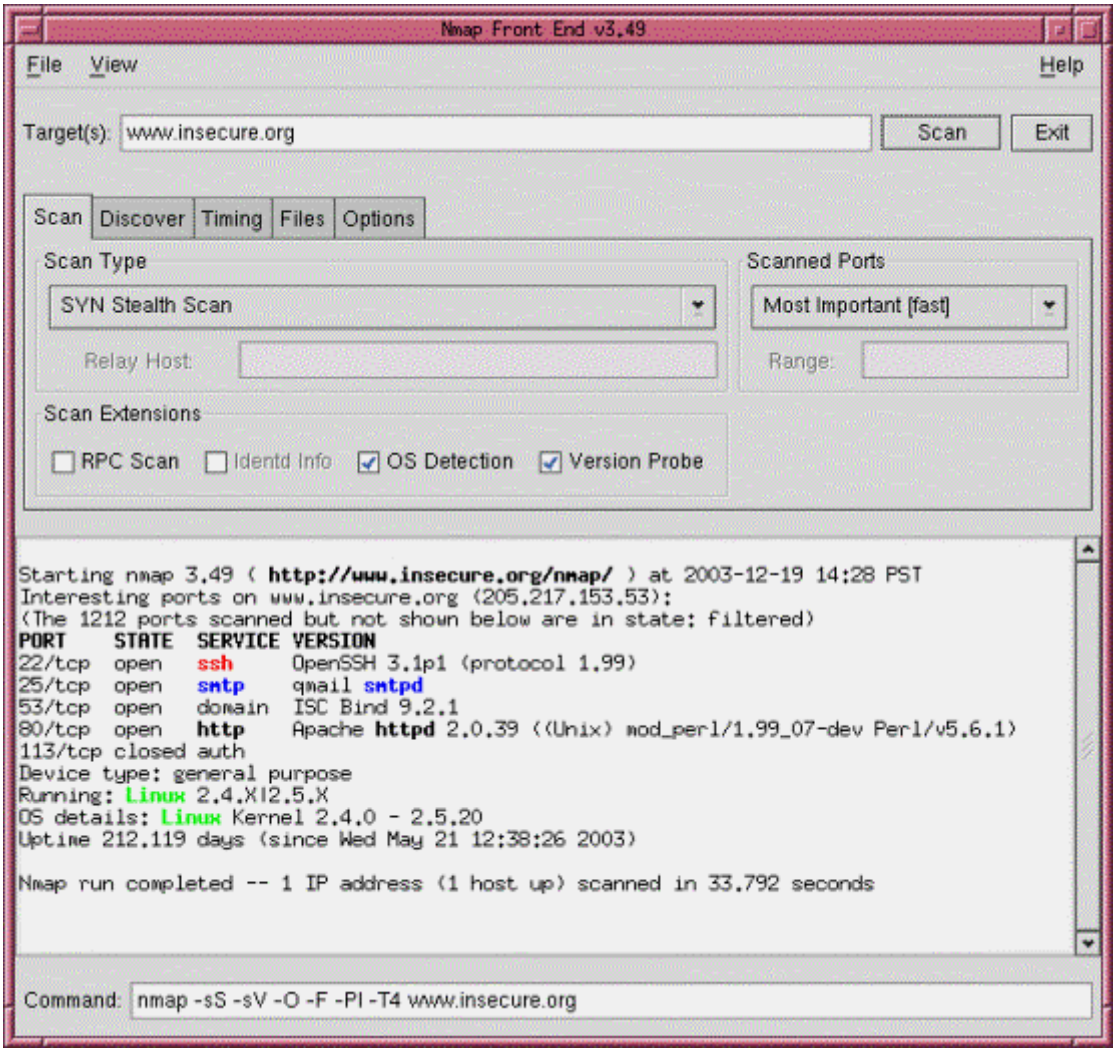


Рис 2 – Зовнішній вигляд утиліти nmap.

Більшість брандмауерів підтримують видалене керування через Інтернет, відкриваючи один або кілька TCP-портів, унікальних для кожного брандмауера. Так, наприклад, Check Point Firewall відкриває 256, 257 й 258 порти, а Microsoft Proxy - 1080. Деякі брандмауери явно повідомляють своє ім'я й версію програмного продукту при підключенні до них по netcat (або telnet), особливо цим грішать Proxy-сервера. Послідовно опитуючи всі вузли, розташовані поперед досліджуваного хосту, на предмет прослуховування характерних для брандмауерів портів, ми в більшості випадків зможе не тільки виявити їхню присутність, але й визначити IP-адреса! Зрозуміло, ці порти можуть бути закриті як на самому брандмауері (правда, не всі брандмауери це дозволяють), так і на попередньому йому маршрутизаторі (але тоді брандмауером буде не можна управляти через Інтернет).

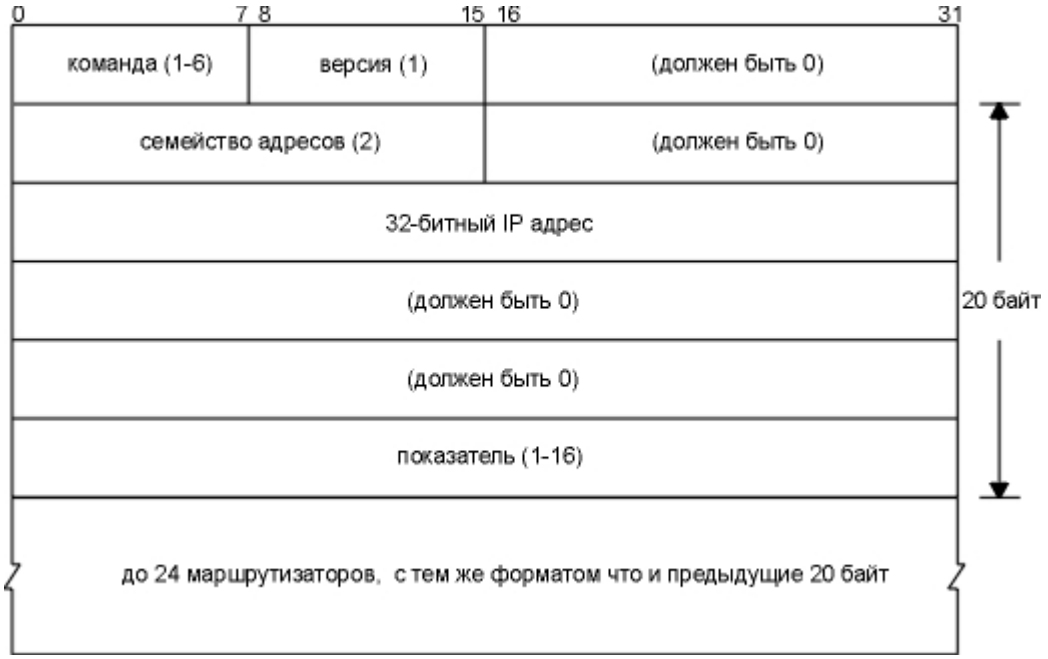


Рис. 3 – Структура IP-пакета.

Сканування й трасування через брандмауер

Пряме трасування через брандмауер найчастіше виявляється неможливим і атакуючий доводяться прибігати до всіляких хитрувань.

Утиліта Firewalk являє собою класичний трассер, що посилає TCP або UDP пакети з таким розрахунком, щоб на вузлі, що випливає безпосередньо за брандмауером, їх TTL звертається в нуль, змушуючи систему генерувати повідомлення ICMP_TIME_EXCEEDED, завдяки чому firewalk упевнено працює навіть там, де штатні засоби вже не справляються.

Будемо виходити з того, що з кожним відправленим IP-пакетом, система збільшує ID на одиницю. З іншого боку, відповідно до специфікації RFC-793, що описує TCP протокол, усякий хост, що одержав сторонній пакет, що не відноситься до встановлених TCP-з'єднань, повинен реагувати на нього посилкою RST. Для реалізації атаки нам знадобитися вилучений вузол, що не обробляє в цей момент ніякого стороннього трафіку й генеруючий передбачувану послідовність ID. У хакерских колах такий вузол називається німим (dump). Виявити німий хост дуже просто - досить лише відправити йому серію IP-пакетів і проаналізувати ID, повернутий у заголовках. Запам'ятаємо ID останнього пакета. Потім, вибравши жертву, що підходить для атаки, відправимо їй SYN-пакет, указавши у зворотній адресі IP вузла. Вузол, що атакує, думаючи, що німий хост хоче встановити з ним TCP-з'єднання, відповість: SYN/ACK. Німий хост, одержавши, сторонній SYN/ACK, поверне RST, збільшуючи свій лічильник ID на одиницю. Відправивши німому хосту ще один IP-пакет, хакер, зрівнявши повернутий ID, зможе довідатися - чи посилав німий хост жертві чи RST-пакет ні. Якщо посилав, виходить, що атакує хост активний і підтверджує установку TCP-з'єднання на заданий порт. При бажанні, хакер може просканувати всі його порти, які цікавлять, не ризикуючи виявитися заміченим, адже дізнатися його IP практично неможливо - сканування здійснюється "руками" німого вузла й з погляду що атакує виглядає як звичайне SYN-сканування.

Припустимо, що німий хост розташовано усередині DMZ, а жертва перебуває усередині корпоративної мережі. Тоді, відправивши німому хосту SYN-пакет від імені жертви, ми зможемо проникнути через брандмауер, оскільки він буде думати, що з ним установлює з'єднання внутрішній хост. А з'єднання цього типу в 99,9% випадках дозволені (якщо їх заборонити, користувачі корпоративної мережі не зможуть працювати

зі своїми власними публічними серверами). Природно, всі маршрутизатори на шляху від хакера до німого хосту, не повинні блокувати пакет з підробленою зворотною адресою, у противному випадку пакет умре задовго до того, як добереться до місця призначення.

Утиліта hping саме й реалізує сценарій сканування даного типу, що робить її основним знаряддям зловмисника для дослідження корпоративних мереж, захищених брандмауером.

Як варіант, хакер може захопити один з вузлів, розташованих усередині DMZ, використовуючи їх як плацдарм для подальших атак.

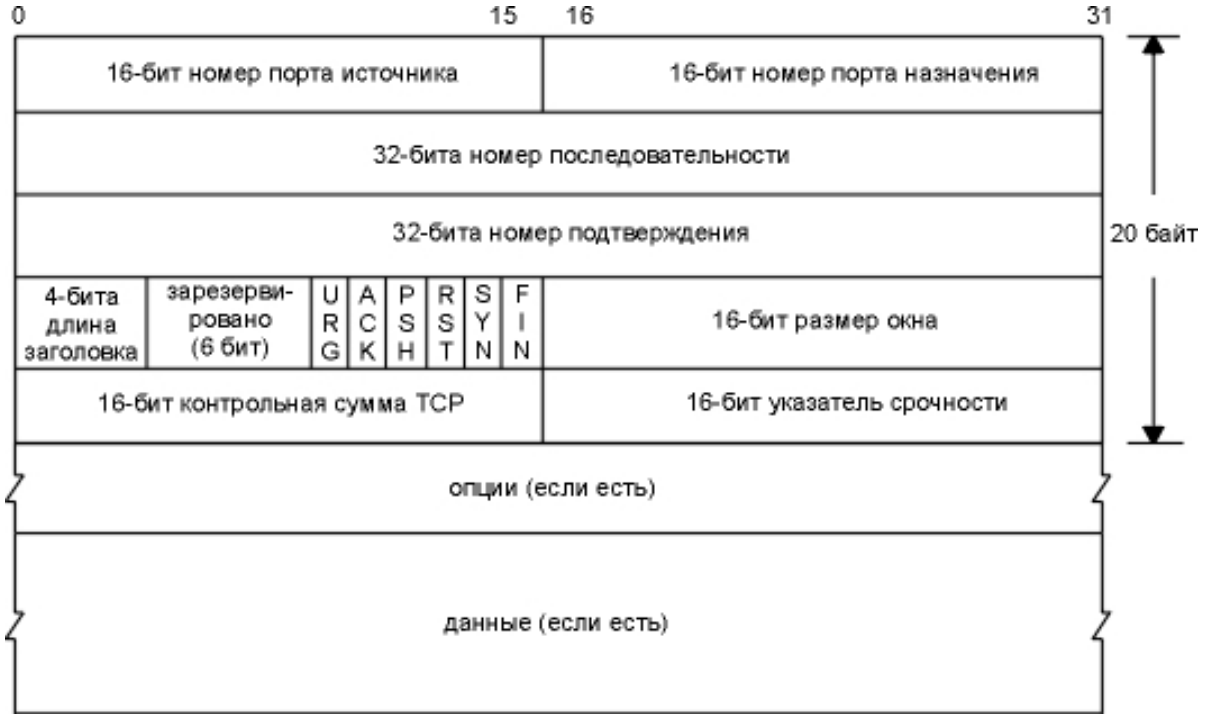


Рис. 4 – Структура TCP-пакета

Проникнення через брандмауер

Збірку фрагментованих TCP-пакетів підтримують тільки самі якісні із брандмауерів, а всі інші аналізують лише перший фрагмент, безперешкодно пропускаючи всі інші. Послявши сильно фрагментований TCP-пакет, "" TCP-заголовок, що розмазується, по декількох IP-пакетах, хакер сховає від брандмауера Acknowledgment Number і він не зможе визначити приналежність TCP-пакета до відповідній йому TCP-сесії. Якщо тільки на брандмауері не активована опція "різати фрагментовані пакети", успіх хакерьскої операції гарантований. Блокування фрагментованих пакетів створює безліч проблем і перешкоджає нормальній роботі мережі. Теоретично можливо блокувати лише пакети із фрагментованим TCP-заголовком, однак, не всякий брандмауер підтримує настільки гнучку політику настроювання. Атаки даного типу, називаються Tiny Fragment Attack і мають надзвичайно потужну проникаючу здатність і тому є улюбленим прийомом всіх хакерів.

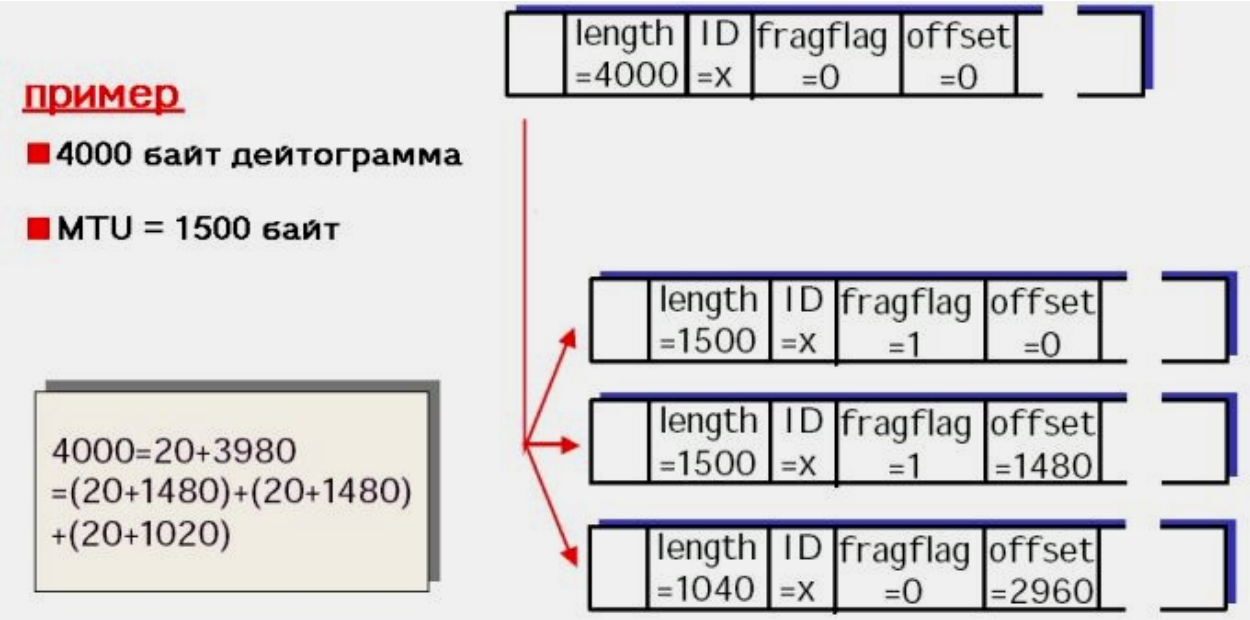


Рис. 5 – Фрагментація TCP-пакетів як спосіб обходу брандмауерів

Атаки з використанням внутрішньої маршрутизації (вона ж - маршрутизація від джерела або source routing) набагато менш актуальні, ми все-таки їх розглянемо. Як відомо IP-протокол дозволяє включати в пакет інформацію про маршрутизації. При відправленні IP-пакета жертві, нав'язане хакером маршрутизація найчастіше ігнорується й траєкторія переміщення пакета визначається виняткова проміжними маршрутизаторами, але відповідні пакети повертаються по маршруті зворотному, зазначеному в IP-заголовку, що створює сприятливі умови для його підміни. Більше спрощений варіант атаки обмежується однією лише підміною IP-адреси відправника, посилаючи пакет від імені одного з їхніх внутрішніх вузлів. Правильно настроєні маршрутизатори (і більшість клонів UNIX) блокують пакети із внутрішньою маршрутизацією. Пакети з підробленими IP-адресами представляють трохи більшу проблему, однак якісний брандмауер дозволяє відсівати і їх.

Таблиці маршрутизації можуть бути динамічно змінені посилкою повідомлення ICMP Redirect, дозволяючи (принаймні, теоретично) направити хакерский трафік в обхід брандмауера (див. також ARP spoofing), однак, у реальному житті такі системи практично ніколи не зустрічаються.

Обхід брандмауеру

Користувачі внутрішньої мережі, захищеної недемократичним брандмауером, серйозно обмежені у своїх можливостях (про неможливість роботи з FTP-серверами в активному режимі ми вже говорили). Так само можуть бути заборонені деякі протоколи й закриті необхідні вам порти. У клінічних випадках адміністратори ведуть списки "чорних" IP-адрес, блокуючи доступ до сайтів "недоцільної" тематики.

Оскільки брандмауери розраховані на захист ззовні, а не зсередини, вирватися через їх захист дуже просто, досить лише скористатися будь-яким підходящим Proxy-сервером, що перебуває в зовнішній мережі й ще не занесеним адміністратором в "чорний список". Зокрема, популярний клієнт ICQ дозволяє обмінюватися повідомленнями не прямо, а через сервер. Існують тисячі серверів, що підтримують роботу ICQ.

Так само, ви можете скористатися протоколом SSH (Secure Shell), споконвічно спроектованим для роботи через брандмауера й підтримуюче шифрування трафіка (на той випадок, якщо брандмауер здумає шукати в ньому "заборонені" слова типу "sex", "hack" і

т.д.). SSH-протокол може працювати по будь-якому доступному порту, наприклад, 80, і тоді з погляду брандмауера все буде виглядати як легальна робота з WEB-сервером. Тим часом, SSH є лише фундаментом для інших протоколів, з яких у першу чергу хотілося б відзначити протокол telnet, що забезпечує взаємодія з видаленими терміналами.

Нарешті, можна скористатися телефонією, прямим модемним підключенням й іншими комунікаційними засобами, що встановлюють з'єднання із провайдером, в обхід брандмауера.

Висновки

Технології побудови брандмауерів не стоять на місці. З кожним днем хакерство стає усе складнішим і складнішим, однак, повністю хакерство не зникне ніколи. Адже на зміну заткнутим дірам приходять інші.

Зауваження:

- Брандмауери піддаються великій кількості DoS атак, таких, наприклад як, луна-шторм або SYN-flood, яким вони в принципі нездатні протистояти;
- Брандмауер це - маршрутизатор, проксі-сервер і система виявлення вторгнень в одному флаконі;
- Брандмауери не захищають від атак, а лише огороджує локальну мережу цегельним забором, через який легко перелізти;
- У більшості випадків через цегельну стіну брандмауера можна пробити ICMP-тунель, обернувши передані дані ICMP-заголовком;
- Брандмауер можна атакувати не тільки ззовні, але й зсередини корпоративної мережі;
- Різні брандмауери по різному реагують на нестандартні TCP-пакети, дозволяючи ідентифікувати себе;
- Брандмауери, що відкривають 53 порт (служба DNS) не тільки на приймачі (наприклад, Check Point Firewall), але й на джерелі, дозволяють хакеру просканувати всю внутрішню мережу;
- Уразливість програмних проксі в загальному випадку невелика й в основному вони атакуються через помилки переповнення буфера;
- Служба DCOM має потребу в широкому діапазоні відкритих портів, що істотно знижує ступінь захищеності системи, відсуваючі брандмауер.