ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС ЗАЩИТЫ РАСЧЕТНЫХ ПРОГРАММ

Добряков Д.Г, Иванов А.Ю., Круглый А.А., Назаренко В.И.
Кафедра ЭВМ ДонГТУ
ai@cs.dgtu.donetsk.ua

Abstract

Dobrykov D.G., Ivanov A.Y.,Krugluy A.A.,Nazarenko V.I. Hardware and software complex for protect calculate program. *The problems of definition of hardware and software for protect of computer is described. Results of designing of two types of toggles and any tricks of program realization is presented. Program modules is realized for Windows system.*

Развитие правовой культуры в обществе, нашедшее свое отражение в статьях 361-363 Уголовного кодекса Украины, не снимает с разработчиков проблем защиты разрабатываемых программных комплексов. Системы защиты компьютерных данных широко распространены [1-5] и находятся в постоянном развитии, стимулирующимся быстрым развитием программного обеспечения и телекоммуникационных систем. Анализ критериев оценки, показателей применимости и классификация [3,4] систем защиты по методу установки, по механизмам защиты, по принципу функционирования приводит к выводу, что наиболее продуктивным для защиты от несанкционированного копирования является комбинированный подход. В соответствии с этим аппаратная часть разработанной службы защиты базируется на электронном ключе для параллельного порта компьютера. Программная часть службы защиты ориентирована на ядро операционных систем семейства Windows NT (Windows 2000, Windows XP), при этом обеспечена совместимость с Windows 9x.

Анализ интерфейсов расширения современных компьютерных систем (COM, LPT, USB, PCI) приводит к выводу о целесообразности построения электронного ключа с логическими уровнями ТТЛ для параллельного интерфейса расширения. Для пассивного электронного ключа, не имеющего больших объемов обмена информацией с компьютером, выбран один из пяти параллельных протоколов стандарта IEEE 1284 – SPP (Standard Parallel Port). Прозрачная структура ключа, при большей сложности и степени аппаратных затрат, сохраняет возможность расширения компьютера. Учтено, что поведение принтера зависит не только от сигналов интерфейса, но и от соответствующих переключателей известных принтеров. Среди всех сигналов управления и статуса только сигналы $\overline{SelectIn}$, $\overline{Strobe}$ и $\overline{Ask}$ полностью соответствуют протоколу SPP при использовании разных принтеров. Из регистрового интерфейса адаптера использованы три регистра, к которым программа обращается по подряд идущим адресам ввода-вывода и которые можно определить из области данных базовой системы ввода-вывода (BIOS) или в системном реестре Windows.

Функциональная схема аппаратной части системы защиты состоит из блока приема данных, блока обработки данных, блока выдачи кода. Входными линиями блока приема являются сигналы шины *Data* стандартного параллельного порта. Блок обработки данных выполняет обработку байтов информации (вычисление необратимой функции), поступающих от блока приема. Блок выдачи кода выполняет выдачу кода, соответствующего результату обработки информации, в параллельный адаптер и использует четыре линии статуса параллельного адаптера: *Busy, Select, PaperEnd* и $\overline{Error}$. Для управления схемой электрон-

ного ключа использован сигнал стандартного параллельного адаптера $\overline{SelectIn}$, который обеспечивает прозрачную работу схемы.

При создании пассивной схемы проявляется отсутствие линии питания у стандартного параллельного порта, что приводит к необходимости получать питание от сигнальных линий, которые на это не рассчитаны и не могут обеспечить большой мощности. Поэтому целесообразно реализовать блоки приема и обработки данных электронного ключа на одной микросхеме, а блок выдачи кода реализовать отдельной микросхемой. Это позволило упростить схему и ипользовать питание от сигнальных линий. Принципиальная схема аппаратной части системы защиты построена на микросхемах серии 74AC (КР1554). Блок выдачи кода построен на базе микросхемы 74AC257N и пропускает на линии статуса параллельного порта статусные линии принтера, при этом схема электронного ключа отключена от линий статуса порта. Если же компьютер устанавливает сигнал $\overline{SelectIn}$ в верхний уровень, то блок коммутирует на линии статуса параллельного адаптера выходы блока обработки данных электронного ключа, что обеспечивает подключение аппаратной части схемы защиты к компьютеру. Таким образом, основным режимом работы электронного ключа является прозрачный режим. Для питания микросхем, составляющих схему электронного ключа, использованы линии $\overline{Strobe}$, $\overline{Init}$, $\overline{AutoFd}$ и $\overline{SelectIn}$, объединенные в один сигнал питания. Объединение линий управления выполняется с помощью диодов Д311А, обеспечивающих падение напряжения на 0,3В. Весь протокол обмена данными с электронным ключом реализуется программным путем.

При разработке принципиальной схемы активного ключа использован микроконтроллер AT89C51 [8] фирмы «ATMEL». В качестве рабочих протоколов обмена выбраны: SPP для передачи данных в направлении от компьютера к ключу, Nibble в обратном направлении. Микроконтроллер содержит внутреннюю Flash-память, в которой хранится управляющая микропрограмма, выполняющая: наблюдение команды входа в механизм защиты и выхода из него, наблюдение команд механизма защиты, выполнение этих команд. Чтение из ключа по стандартному протоколу Nibble выполняется стандартно. При записи информации в ключ данные передаются последовательно через линию #AutoFd, начиная со старшего разряда. Цикл записи начинается с установки на линию #AutoFd очередного бита информации, потом компьютер должен перевести линию #Strobe в низкий уровень – строб. Если контроллер будет готов принять бит информации, он должен установить в низкий уровень линию #Ack – подтверждение. После получения сигнала подтверждения компьютер должен снять строб (#Strobe перевести в высокий уровень), а контроллер должен откликнуться, сняв сигнал подтверждения. Такая система надежно исключает потерю передаваемых данных. Кроме того, подтверждения позволяют исключить возможную рассинхронизацию приемника и передатчика при неминуемых различиях в скоростях. Все функции, которые реализует микроконтроллер, кроме функции генератора Парка-Миллера, выполняются тривиальным способом и детального рассмотрения не требуют. Результат функции, как и входной параметр, содержит 32 разряда. Так как микроконтроллер может обрабатывать лишь 8-ми разрядные данные, то непосредственное выполнение этой функции невозможно. Для выхода из этого положения предложено выполнять умножение по аналогии с умножением двоичных чисел (алгоритм В). Умножение выполняется побайтно, а затем частичные произведения суммируются, предварительно сдвинутые на 8, 16 или 24 разряда соответственно месту множимого. При применении кварцевого резонатора в 24 Мгц все таймеры-счетчики микроконтроллера увеличивают на единицу свое содержимое каждую половину микросекунды. Прерывание от таймеров-счетчиков наступает при переходе через нуль. Таким образом, константа пересчета для интервала 20 мс равна 63BFh.

Программная часть системы защиты реализует следующие функции: взаимодействие с электронным ключом, привязка программного обеспечения к компьютеру, комплекс-

ная защита от несанкционированной отладки. Первая функция проверяет присутствие в системе электронного ключа и на основе результата проверки принимает решение об авторизации использования программного обеспечения. Третья функция служит для предотвращения попыток отключения двух предшествующих функций и использует специфические методы и свойства операционных систем для нахождения и устранения попыток вмешательства в функционирование программы.

Программная часть комплекса базируется на потоковом принципе функционирования, то есть все функции модуля оформлены отдельными потоками операционной системы. Такое построение позволило контролировать авторизацию использования программного обеспечения на протяжении всего времени его функционирования. Электронный ключ проверяется не только при запуске программы, но и всего периода функционирования через определенный интервал времени, что не дает пользователю возможности использовать один электронный ключ с несколькими копиями программного обеспечения. Потоковый принцип построения системы защиты также позволяет создать распределенную модель системы защиты (все потоки запускается в разные моменты времени) и значительно усложнить всю систему защиты благодаря параллельному и непрерывному функционированию всех частей системы.

Модуль взаимодействия с электронным ключом предназначен для реализации интерфейса с аппаратной частью системы защиты и включает генератор необратимой функции, модуль проверки электронного ключа, драйвера взаимодействия с параллельным портом. Особенностью генератора есть то, что он работает как отдельный поток операционной системы и выполняет вычисления в течении всего времени работы программы. Для поиска параллельного адаптера по всем возможным базовым адресам используется стандартный механизм записи и чтения по базовому адресу порта. Если операция была выполнена успешно, результаты записи и чтения совпали, то предполагается, что по этому базовому адресу присутствует параллельный порт, и выполняется проверка ключа. Последовательности действий реализующих протокол работы с электронным ключом завершается проверкой результата. Результат от электронного ключа снимается из регистра статуса параллельного адаптера. Если рассчитанный и полученный от ключа результаты совпали, электронный ключ считается присутствующим в системе и поиск прекращается.

Работа с электронным ключом требует взаимодействия с параллельным портом на уровне портов ввода-вывода. Под управлением операционной системы типа Windows прямой доступ к портам ввода-вывода невозможен и попытки обращения к ним или игнорируются или вызовут исключение. Эта проблема преодолевается путем разработки драйверов, которые выполняются на уровне привилегий ядра операционной системы, и могут непосредственно обращаться к портам ввода-вывода, выполняя привилегированные команды процессора. Поскольку схема взаимодействия драйверов с операционной системой отличается в Windows NT и Windows 9x [9 -12], то необходимо наличие двух драйверов, по одному для каждого типа операционной системы. Для поддержки разработанных двух таких драйверов была создана динамически загружаемая библиотека. Взаимодействие с портами ввода-вывода в драйверах реализовано разными способами. В операционной системе Windows 9x для доступа к портам ввода-вывода необходимо обратиться к драйверу, который непосредственно выполнит операции обмена с портом и возвратит результат своей работы. А в операционной системе Windows NT драйвер лишь открывает программе прямой доступ к портам ввода-вывода, а программа сама непосредственно читает и записывает данные в порт. В операционной системе Windows NT использована карта разрешения прямого доступа к портам ввода-вывода. Эта карта представляет собой битовый массив, каждый бит которого разрешает (нулевое значение) или запрещает (единичное значение) прямой доступ к портам ввода-вывода. Для того, чтобы программа, выполняющаяся на третьем уровне привилегий, имела возможность обращаться к портам ввода-вывода, для нее от-

крывается доступ к карте разрешения и заносятся соответствующие данные. Для выполнения этого использованы функции *Ke386SetIoAccessMap(int,IOPM)* и *Ke386IoSetAccessProcess(PEPROCESS, int)*. Эти функции включены в состав файла NTOSKRNL.EXE, но не документированы. Для открытия доступа к портам ввода-вывода драйверу передается команда *IOCTL_WINIO_ENABLEDIRECTIO*. Если прямой доступ к портам ввода-вывода уже не требуется, то выполняется команда *IOCTL_WINIO_DISABLEDIRECTIO*, закрывающая доступ к портам ввода-вывода.

Таким образом, модуль взаимодействия с электронным ключом реализует все функции, необходимые для создания интерфейса с аппаратной частью системы защиты.

Модуль привязки программного обеспечения к компьютеру выполняет функцию ограничения количества используемых покупателем копий программы. Для этого во время первого запуска программного обеспечения выполняется установка в операционной системе и в самой программе признаков, отсутствие которых во время дальнейших запусков не разрешает функционирования программы. В модуле реализованы два уровня привязки: регистрация программы в системном реестре и модификация файла главного модуля программного обеспечения.

Любая программа, которая защищается от несанкционированного использования, содержит в своем составе модуль регистрации. Разработанный модуль поддерживает два режима работы: режим временного использования (здесь 30 суток) и режим постоянного пользования. Им можно воспользоваться лишь при наличии регистрационного номера электронного ключа и самого ключа. Регистрация программного обеспечения в системном реестре выполняется после того, как пользователь предоставит индивидуальный серийный номер данной копии; в дальнейшем ключ проверяется во время каждого запуска программного обеспечения. При этом сознательно реализован очень простой механизм проверки серийного номера и регистрации программного обеспечения, для того чтобы отвлечь внимание взломщика от второго уровня привязки. Второй уровень привязки выполняет настройку программного обеспечения на индивидуальные особенности компьютера. Настройка программного обеспечения выполняется путем модификации файла библиотеки защиты главного модуля. Уникальные характеристики компьютера считываются из области данных BIOS компьютера. При этом, если в Windows 9x дату издания можно получить непосредственно из программы пользователя, то в операционных системах семейства Windows NT для этого нужно иметь привилегии ядра операционной системы. С целью унификации и для запутывания взломщика чтение BIOS в операционных системах семейства Windows 9x также выполняется из уровня привилегий ядра операционной системы. Собственный обработчик прерывания устанавливается с помощью модификации системной таблицы дескрипторов обработчиков прерываний IDT (IDT – Interrupt Description Table). Хотя существуют также другие методы получения привилегий ядра операционной системы (например, с помощью глобальной таблицы дескрипторов GDT), но они являются более сложными и могут вызвать нестабильность операционной системы. Функция чтения BIOS вызывается как программное прерывание int 5 и поэтому выполняет все свои действия с привилегиями ядра операционной системы. Чтение BIOS в Windows NT выполняется иначе. В этом случае блок данных BIOS не отображается в виртуальное адресное пространство программы, поэтому прямое чтение информации из него невозможно. Для того чтобы прочитать данные, нужно в ручном режиме выполнить отображение требуемого блока физической памяти в виртуальное адресное пространство процесса. Поскольку обычная программа таких действий выполнять не может из-за отсутствия привилегий ядра системы, то для решения этой проблемы был разработан специальный драйвер, работающий в режиме ядра системы. Этот драйвер предоставляет модулю привязки программный интерфейс в виде функций *InstallDriver*, *UninstallDriver* и *GetBiosInfo*.

Функция *InstallDriver* предназначена для установки драйвера в системе. Для этого создается системный сервис с именем драйвера – *InfoDriver* и запускается работа этого сервиса. После создания сервиса операционная система загружает файл драйвера и передает выполнение функции *DriverEntry*, которая выполняет все действия, необходимые для инициализации драйвера. Функция выполняет создание символической ссылки на драйвер, по которой к нему будет обращаться программа, а также выполняет запись в объект ядра драйвера адреса функций *InfoDrvDispatch*, которая будет заниматься обработкой сообщений драйверу. После инсталляции драйвера программа может обращаться к драйверу с помощью функции *GetBiosInfo*. Эта функция выполняет открытие объекта драйвера по символической ссылке с помощью функции *OpenDevice* и посылает ему сообщение о необходимости чтения BIOS с использованием системного сервиса *DeviceIoControl*. Объект драйвера, получив сообщение, вызовет функцию *InfoDrvDispatch*, которая проверяет правильность сообщения, обеспечивает присоединение буфера возврата считанного значения к области данных драйвера и вызывает функцию *InfoDrvFastIoDeviceControl*. Эта функция служит для реакции на сообщение драйверу. После вызова функция *InfoDrvFastIoDeviceControl* выполняет отображение блока физической памяти с областью данных BIOS в виртуальное адресное пространство драйвера, чтение из него информации BIOS и отключение блока физической памяти от виртуального адресного пространства. Считанные данные возвращаются в модуль привязки с помощью системного буфера. Для присоединения блока физической памяти функция *InfoDrvFastIoDeviceControl* использует функции ядра операционной системы *ZwOpenSection*, *ObReferenceObjectByHandle*, *ZwMapViewOfSection*. Эти функции отображают блок физической памяти в виртуальное адресное пространство драйвера, начиная с нужного адреса и создавая блок нужной длины. На базе значений, возвращаемых этими функциями, выполняется расчет виртуального адреса блока данных BIOS, по которому будет выполняться чтение даты издания. Функция *UninstallDriver* выполняет удаление драйвера как сервиса операционной системы. При удалении выполняется уничтожение объекта драйвера и удаление символической ссылки на драйвер из операционной системы.

Сохранение данных происходит путем модификации файла библиотеки защиты программного обеспечения. Для этого в файле библиотеки защиты до первого запуска декларируется переменная со специфическими полями, адрес расположения которой в файле известен заранее. Модуль защиты передает зашифрованные данные дочернему процессу, запускаемому после окончания работы программного обеспечения и выполняющего запись этих данных по нужному адресу в файл библиотеки защиты программного обеспечения. Необходимость дочернего процесса определяется тем, что операционная система запрещает модификацию файлов программы во время ее выполнения. Поскольку привязка производится лишь во время первого запуска программного обеспечения, то все действия проходят незаметно для пользователя, и вместе с модулем противодействия несанкционированной отладки их трудно определить. Проверка привязки программного обеспечения к компьютеру выполняется в отдельном потоке программы и проходит таким же образом, как и начальное получение данных BIOS. Следует отметить, что во время проверки не выполняется дополнительного чтения из файла, поскольку переменная уже загружена в оперативную память. Для синхронизации потоков начальной привязки и ее проверки во время первого запуска программного обеспечения использован объект синхронизации мьютекс [9,10]. Функции модуля привязки выполняют все действия, необходимые для ограничения количества копий программного обеспечения у пользователя. Они могут работать как отдельные потоки операционной системы, так и как обычные функции программы, но рекомендованным является их поточное использование.

Несанкционированная отладка является наиболее распространенным методом преодоления систем защиты программного обеспечения. Поэтому для создания комплексной системы защиты программного обеспечения реализован сложный механизм защиты. Мо-

дуль защиты от несанкционированной отладки состоит из трех частей: проверка присутствия в контексте операционной системы программ мониторинга и отладки, проверка отладки программного обеспечения в операционных системах типа Windows NT, проверка времени выполнения контрольных участков кода программы.

Для того чтобы проводить несанкционированную отладку программного обеспечения, взломщику необходимо сначала отследить, какие действия выполняет система защиты программы. Для этого существует наиболее популярный набор утилит компании *www.sysinternals.com*, который состоит из следующих программ: FileMon – мониторинг операций с файлами; DiskMon – мониторинг дисковых операций; PortMon – мониторинг операций с портами ввода-вывода; RegMon – мониторинг операций с системным реестром. Так как эти программы в своей работе используют специфические драйверы, то для мониторинга системных операций нужно иметь уровень привилегий ядра операционной системы. Поскольку имена драйверов этих программ известны, а главное они не изменяются разработчиком, то по известному имени драйвера можно узнать, работает ли в системе та или иная программа мониторинга. Для этого используется системная функция *CreateFile*. Если драйвер программы мониторинга работы с файлами FileMon присутствует в системе, то функция возвратит реальный дескриптор этого драйвера. Однако для Windows NT существует одна особенность: если программа во время проверки активна, то система не возвратит реальный дескриптор. Для устранения этой особенности использован контроль ошибок, возникающих во время вызова функции *CreateFile*. Если возникла ошибка из-за попытки одновременного использования драйвера, то в системе присутствует программа мониторинга, и она в это время находится в активном состоянии. С помощью этого метода также отслеживается присутствие в системе программы отладки *SoftIce*. Существуют и другие способы отслеживания программ мониторинга и отладки в системе. Один из них - проверка всех присутствующих в системе процессов. Но производители этих программ имеют привычку периодически изменять названия процессов, вследствие чего этот способ не может гарантировать полной достоверности определения программ мониторинга и отладки. Для операционных систем Windows NT проверка выполнения программ под управлением программы отладки базируется на использовании системной функции *IsDebuggerPresent*, позволяющей установить из программы, загружена ли она в контексте программы отладки. При этом особенностью является то, что используется системная функция, обеспечивающая нахождение всех возможных программ отладки, и от которой ни одна программа отладки не имеет защиты, в связи с использованием этой функцией специфических сервисов ядра операционной системы. Также положительным свойством является то, что этот способ недостаточно документирован, так как функция *IsDebuggerPresent* стандартно не экспортируется из библиотеки функций ядра операционной системы, ее дополнительно нужно импортировать во время выполнения программы в ручном режиме.

Для противодействия программам отладки проверяется существование точек останова на те функции WIN API, которые использует защищаемая программа. Недостатком проверки первого байта функции WIN API на равенство кода 0xcc есть то, что для достижения результата также используются функции WIN API. Этого недостатка лишен метод, базирующийся на использовании таблицы импорта [10]. Если программа вызовется на выполнение, загрузчик программы заполняет эту таблицу истинными значениями. Итак, во время выполнения программы ее таблица импорта содержит адреса всех функций WIN API, которые она использует в своей работе. В PE-файле, в случае вызова функции из другого модуля (например, GetMessage() из USER3DLL), инструкция CALL, которая сгенерирована компилятором, не передает управления непосредственно данной функции в DLL. Вместо этого инструкция CALL передает управление команде JMP DWORD PTR[XXXXXXXX], что также находится в секции .text (секция кода этой программы). Команда JMP перескакивает к адресу, который сохраняется в двойном слове в секции .idata (таблица импорта). Это

двойное слово в секции .idata содержит истинный адрес точки входа функции операционной системы. Так генерирует код компилятор Borland C++, а при использовании компилятора Visual C++ вызов функции будет иметь вид CALL DWORD PTR[XXXXXXXX].

Действенным средством против дизассемблирования является криптографическая защита данных и программ [13 -15]. При этом программа при функционировании выполняет раскодирование самой себя. Создание самомодифицирующегося кода требует знания недокументированных деталей архитектуры Windows, но одинаково реализованных на всех Windows-платформах и активно используемых компилятором Visual C++ от Microsoft. Для адресации 4 ГБ виртуальной памяти, выделенной в распоряжение процесса [9-11], Windows использует два селектора; один из них загружается в сегментный регистр CS, а другой - в регистры DS, ES и SS. Оба селектора ссылаются на одинаковый базовый адрес памяти (равняется нулю), и имеют лимит в 4 ГБ. Фактически существует всего один сегмент, который вмещает у себя и код, и данные, и стек процесса. Благодаря этому передача управления коду, расположенному в стеке, осуществляется близким (near) вызовом или переходом, и для доступа к содержимому стека использование префикса "SS" совсем необязательно. Несмотря на то, что значение регистра CS не равняется значению регистров DS, ES и SS, команды MOV dest,CS:[src]; MOV dest,DS:[src] и MOV dest,SS:[src] в действительности обращаются к одной и той же ячейке памяти. Отличия между регионами кода, стека и данных состоят в атрибутах принадлежащих им страниц: страницы кода допускают чтение и выполнение, страницы данных - чтение и запись, а стека - чтение, запись и выполнение одновременно. Кроме этого, каждая страница имеет специальный флаг, который определяет уровень привилегий, необходимых для доступа к этой странице. Манипулировать атрибутами страниц, равно как и ассоциировать страницы с линейными адресами, может только операционная система (или код, который выполняется в нулевом кольце). Если нужно изменить определенное количество байт своего (или чужого) процесса, простейший способ сделать это – вызвать функцию WriteProcessMemory. Она разрешает модифицировать страницы памяти с неустановленным флагом супервизора. При этом критические структуры данных ОС (например, page directory или page table) остаются доступны лишь из нулевого кольца. Поэтому эта функция не представляет никакой угрозы для безопасности системы, и успешно вызовется независимо от уровня привилегий пользователя. Процесс, в память которого происходит запись, должнен быть предварительно открыт функцией OpenProcess с атрибутами доступа "PROCESS_VM_OPERATION" и "PROCESS_VM_WRITE". Однако, использовать WriteProcessMemory в защитах – представляется несколько наивным. Опытный взломщик сразу найдет подвох, заметив эту функцию в таблице импорта. Другое ограничение WriteProcessMemory - невозможность создания новых страниц; ей доступны уже существующие страницы. Так как в динамической памяти выполнения кода запрещено, то использовано выполнение кода в стеке. Выполнение кода в стеке разрешено, поскольку выполняемый стек необходим многим программам (в том числе и самой ОС) для выполнения некоторых системных функций. Такое решение устраняет оба недостатка функции WriteProcessMemory. Во-первых, обнаружить и отследить команды, которые модифицируют заранее неизвестную ячейку памяти, тяжело и придется провести кропотливый анализ кода защиты; во-вторых, приложение в любой момент может выделить столько стековой памяти, сколько ему понадобится, а потом, при исчезновении потребности ее освободить. Основное преимущество использования стека в том, что код функции, которая исполняется в стеке, можно прямо "на лету" изменять (например, расшифровывать). После этого из исходного текста программы функцию ParkMiller можно удалить, разместив ее зашифрованный текст, например, в строковой переменной. В нужный момент он будет расшифрован, скопирован в локальный буфер и вызван для выполнения.

Для завершенности комплексного характера модуля защиты от несанкционированной отладки использована проверка времени выполнения контрольных участков кода про-

граммы. Проверка времени выполнения контрольных участков кода производятся в отдельном потоке операционной системы, что усложняет систему защиты и дает возможность определить программу отладки, в то время как с ее помощью отслеживается другая функция программного обеспечения.

При практических испытаниях функции системы защиты запускались из главного защищаемого модуля обеспечения, как в потоковом, так и в обычном виде. Для проверки работоспособности системы защиты предпринимались попытки использовать службу без электронного ключа, а также определения механизма привязки программы к компьютеру. Для этого использовались перечисленные программы мониторинга. Механизм регистрации программы в системном реестре удалось установить, но механизм привязки программы к индивидуальным характеристикам компьютера установить в приемлемые сроки не удалось. Также за приемлемый период не удалось установить механизм взаимодействия программы с электронным ключом.

Таким образом, разработанный комплекс защиты показал свою полную работоспособность. Благодаря организации в виде отдельного модуля и реализации комплексного подхода к защите программного обеспечения от несанкционированного использования и копирования, система защиты может быть использована для создания служб защиты программных продуктов.

Литература

1. *Защита информационных систем - http://kiev-security.da.ru.*
2. *Материалы сайта http:\www.pilorama.nm.ru..*
3. *Черней Г. А., Охрименко С. А., Ляху Ф. С. Безопасность автоматизированных информационных систем. – Кишинев.: Ruxanda, 1996. – 250с.*
4. *Середа С. А. Оценка эффективности систем защиты программного обеспечения. – http://infocity.kiev.ua*
5. *Середа С. А. Программно-аппаратные системы защиты программного обеспечения. – http://cie.ase.md/~sereda.*
6. *Груздев С. Электронные ключи. – Еженедельник «Компьютерное обозрение» №20, 29 мая 1996 г.*
7. *Материалы сайта http:\www.aladdin.ru*
8. *Документация фирмы ATMEL на микроконтроллер AT89C51 (http:\\www.atmel.com).*
9. *Рихтер Дж. Windows для профессионалов / Пер. с англ. – 4-е изд. – СПб.: Питер, 2001. – 752 с., ил.*
10. *Microsoft Developer Network – http://msdn.microsoft.com*
11. *Материалы сайта http://www.software-engineer.org/*
12. *Материалы сайта http://www.sysinternals.com/ntw2k*
13. *Молотков С.В., Зегжда Д.П., Мазничка Ю.И., Петров В.А. Информационная безопасность. Защита информации от компьютерных вирусов в сетях ЭВМ. М.: МИФИ, 1993 г. - 68 стр*
14. *Материалы сайта http://www.fssr.ru*
15. *"Введение в криптографию" под редакцией В.В.Ященко - http://www.cryptography.ru*

Поступила в редакційну колегію 01.02.2002 р.