АНАЛИЗ РИСКА ПОЯВЛЕНИЯ УЯЗВИМОСТЕЙ «ПЕРЕПОЛНЕНИЯ БУФЕРА»

Балынский С.Ю., Губенко Н.Е.
Донецкий национальный технический университет

Как показывает анализ в нстастоящее время самым распространенным способом получения привилегированного доступа к компьютеру являеться эксплоитинг (exploiting). При этом доминирующими при при удаленных/локальных атаках являються следующие: «переполнение буфера» (buffer overflow), «возврат на функцию» (Return-To-Function), «формат строки» (Format String), «переполнение кучи» (Heap Overflow)[1]. Bugtraq показывает, что примерно 2/3 респондентов считает переполнение буфера основной причиной нарушения сетевой безопасности[2]. Очевидно, что эффективное решение данной проблемы позволит исключить большую долю самых серьезных угроз компьютерной безопасности.

Основа атак с использованием эксплоитинга – принцип, при котором злоумышленник получает привилегии и права процесса. Таким образом, менее привилегированный пользователь или процесс, который взаимодействует с данной программой, может использовать ее права в своих целях. Использование уязвимостей подразумевает изменение хода выполнения привилегированной программы. Причиной же появления таких уязвимостей является, в первую очередь, сложность современных компьютерных систем, которые неизбежно влекут за собой ошибки проектирования и реализации проекта.

Анализ языка Си показывает, что ошибки переполнения – это фундаментальные программистские ошибки, которые чрезвычайно трудно отслеживать. К тому же поддержка массивов реализована лишь частично и работа с ними требует чрезвычайной аккуратности и внимания со стороны программиста: средства автоматического контроля выхода за границы отсутствуют, возможность определения количества элементов массива по указателю отсутствует, строки должны завершаться нулем. Корректную проверку аргументов невозможно осуществить в принципе! Рассмотрим функцию, определяющую длину переданной ей строки, и посимвольно читающую эту строку до встречи с завершающим ее нулем. А если завершающего нуля на конце не окажется? Тогда функция вылетит за пределы утвержденного блока памяти и пойдет далле по оперативной памяти! В лучшем случае это закончится выбросом исключения. В худшем – доступом к конфиденциальным данным. Можно, конечно, передать максимальную длину строкового буфера с отдельным аргументом, но кто поручиться, что она верна? Ведь этот аргумент приходится формировать вручную и, следовательно, он не застрахован от ошибок. Поэтому вызываемой функции ничего не остается как довериться корректности передаваемых ей аргументов, а если так – то какие проверки реально возможны?!

Ситуация несколько улучшается с переходом к Си++, так как там реализована подержка динамических массивов, но выполняеться крайне медленно и не эффективно для больших объемов информации.

Неправильно думать, что «переполнение буфера» это всего лишь попытка поместить большее количество данных в буфер, чем он способен обработать.

Переполнение при записи приводит к затиранию, а, следовательно, искажению одной или нескольких переменных (включая служебные переменные, внедряемые компилятором, такие, например, как адреса возврата или указатели this), нарушая тем

самым нормальный ход выполнения программы и вызывая одно из следующих последствий:

- нет никаких последний;
- программа выдает неверные данные;
- программа "вылетает", зависает или аварийно завершается с сообщением об ошибке;
- программа изменяет логику своего поведения, выполняя незапланированные действия.

Переполнение при чтении менее опасно, так как приводит только к потенциальной возможности доступа к конфиденциальным данным (например, паролям или идентификаторам TCP/IP соединения).

Наибольшую угрозу для безопасности системы представляют именно указатели, поскольку они позволяют атакующему осуществлять запись в произвольные ячейки памяти или передавать управление по произвольным адресам, например, на начало самого переполняющегося буфера, в котором расположен машинный код, специально подготовленный злоумышленником, и обычно называемый shell-кодом. Буферы, располагающиеся за концом переполняющегося буфера, могут хранить некоторую конфиденциальную информацию (например, пароли). Раскрытие чужих паролей, равно как и навязывание, атакуемой программе своего пароля – вполне типичное поведение для атакующего. Скалярные переменные могут хранить индексы (и тогда они фактически приравниваются к указателям), флаги, определяющие логику поведения программы (в том числе и отладочные люки, оставленные разработчиком) и прочую информацию.

В зависимости от своего местоположения буфера делаться на три независимые категории: а) локальные буфера, расположенные в стеке и часто называемые автоматическими переменными; б) статичные буфера, расположенные в секции (сегменте) данных; в) динамические буфера, расположенные в куче. Подробный анализ особенностей переполнения буферов этих типов приводиться в докладе.

Проблема переполнения буфера являеться сложной как для атакующих, так и для защищающихся. Приведем причины, по которым вопрос о переполнении буфера остается актуальным:

- все разработанные методики борьбы с переполняющимися буферами снижают производительность, но не исключают возможность переполнения полностью;
- межсетевые экраны отсекают лишь самых примитивнейших из червей, загружающих свой хвост через отдельное TCP/IP соединение, отследить же передачу данных в контексте уже установленных TCP/IP соединений никакой межсетевой экран не в силах;
- не существуют никаких надежных методик автоматического (или хотя бы полуавтоматического) поиска переполняющихся буферов, дающих удовлетворительный результат и по-настоящему крупные дыры не обнаруживаются целенаправленным поиском.

К методике защиты от переполнения можно отнести решение Microsoft. DEP – запрет выполнения кода в стеке/куче. Это не гарантирует полной защиты от переполнения, но принципиально повышает надежность программы.

Литература

1. http://www.securitylab.ru/analytics/216312.php (05.11.2005)
2. http://www.security.nnov.ru/articles/bo.asp (11.11.2005)