**Лекция 10. Обратная разработка. Бинарные уязвимости.**

В данной лекции рассматриваются вопросы бинарных уязвимостей.

Под бинарными уязвимостями подразумеваются уязвимости, содержащиеся в исполняемом (скомпилированном) файле. Критичность таких уязвимостей переоценить довольно сложно. Например, если такая уязвимость будет найдена в определённой версии популярного ПО – то она распространяется на все копии этого продукта. Можно просмотреть отличия от уязвимостей веб-приложений. Уязвимость, найденная в каком-либо конкретном веб-приложении не будет распространена на все остальные.

Список типовых бинарных уязвимостей представлен ниже:

1) Переполнение буфера (стек, куча)

2) Уязвимость форматной строки

3) Целочисленное переполнение

4) Use after free

5) RCE

Кратко рассмотрим некоторые из них.

**1. Переполнение буфера.**

Переполнение буфера обычно возникает из-за неправильной работы с данными, полученными извне, и памятью, при отсутствии жесткой защиты со стороны подсистемы программирования (компилятор или интерпретатор) и операционной системы. В результате переполнения могут быть испорчены данные, расположенные следом за буфером (или перед ним).

Переполнение буфера является одним из наиболее популярных способов взлома компьютерных систем, так как большинство языков высокого уровня используют технологию стекового кадра — размещение данных в стеке процесса, смешивая данные программы с управляющими данными (в том числе адреса начала стекового кадра и адреса возврата из исполняемой функции).

Переполнение буфера может вызывать аварийное завершение или зависание программы, ведущее к отказу обслуживания (denial of service, DoS). Отдельные виды переполнений, например, переполнение в стековом кадре, позволяют злоумышленнику загрузить и выполнить произвольный машинный код от имени программы и с правами учетной записи, от которой она выполняется.

Программа, которая использует уязвимость для разрушения защиты другой программы, называется эксплойтом. Наибольшую опасность представляют эксплойты, предназначеные для получения доступа к уровню суперпользователя или, другими словами, повышения привилегий. Эксплойт переполнения буфера достигает этого путём передачи программе специально изготовленных входных данных. Такие данные переполняют выделенный буфер и изменяют данные, которые следуют за этим буфером в памяти.

Представим гипотетическую программу системного администрирования, которая исполняется с привилегиями суперпользователя — к примеру, изменение паролей пользователей. Если программа не проверяет длину введённого нового пароля, то любые данные, длина которых превышает размер выделенного для их хранения буфера, будут просто записаны поверх того, что находилось после буфера. Злоумышленник может вставить в эту область памяти инструкции на машинном языке, например, шелл-код, выполняющие любые действия с привилегиями суперпользователя — добавление и удаление учётных записей пользователей, изменение паролей, изменение или удаление файлов и т. д. Если исполнение в этой области памяти разрешено и в дальнейшем программа передаст в неё управление, система исполнит находящийся там машинный код злоумышленника.

Правильно написанные программы должны проверять длину входных данных, чтобы убедиться, что они не больше, чем выделенный буфер данных. Однако программисты часто забывают об этом. В случае если буфер расположен в стеке и стек «растёт вниз» (например в архитектуре x86), то с помощью переполнения буфера можно изменить адрес возврата выполняемой функции, так как адрес возврата расположен после буфера, выделенного выполняемой функцией. Тем самым есть возможность выполнить произвольный участок машинного кода в адресном пространстве процесса. Использовать переполнение буфера для искажения адреса возврата возможно даже если стек «растёт вверх» (в этом случае адрес возврата обычно находятся перед буфером).

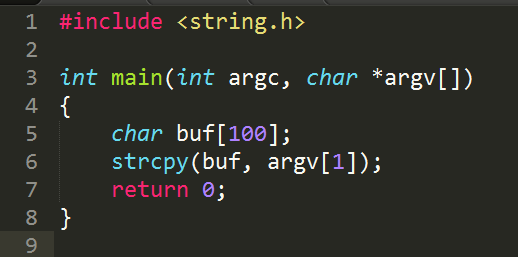
Даже опытным программистам бывает трудно определить, насколько то или иное переполнение буфера может быть уязвимостью. Это требует глубоких знаний об архитектуре компьютера и о целевой программе. Было показано, что даже настолько малые переполнения, как запись одного байта за пределами буфера, могут представлять собой уязвимости.

Переполнения буфера широко распространены в программах, написанных на относительно низкоуровневых языках программирования, таких как язык ассемблера, Си и C++, которые требуют от программиста самостоятельного управления размером выделяемой памяти. Устранение ошибок переполнения буфера до сих пор является слабо автоматизированным процессом. Системы формальной верификации программ не очень эффективны при современных языках программирования.

Многие языки программирования, например, Perl, Python, Java и Ada, управляют выделением памяти автоматически, что делает ошибки, связанные с переполнением буфера, маловероятными или невозможными. Perl для избежания переполнений буфера обеспечивает автоматическое изменение размера массивов. Однако системы времени выполнения и библиотеки для таких языков всё равно могут быть подвержены переполнениям буфера, вследствие возможных внутренних ошибок в реализации этих систем проверки. В Windows доступны некоторые программные и аппаратно-программные решения, которые предотвращают выполнение кода за пределами переполненного буфера, если такое переполнение было осуществлено. Среди этих решений — DEP в Windows XP SP2.

В гарвардской архитектуре исполняемый код хранится отдельно от данных, что делает подобные атаки практически невозможными.

**ПРИМЕР КОДА.**



Используется небезопасная функция strcpy, которая позволяет записать больше данных, чем вмещает выделенный под них массив. Если запустить данную программу в системе Windows с аргументом, длина которого превышает 100 байт, скорее всего, работа программы будет аварийно завершена, а пользователь получит сообщение об ошибке.

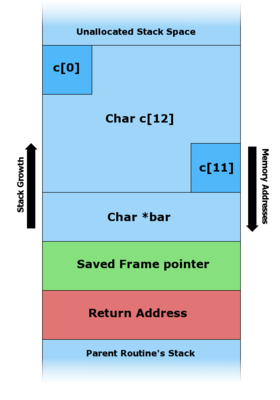
Следующая программа не подвержена данной уязвимости:



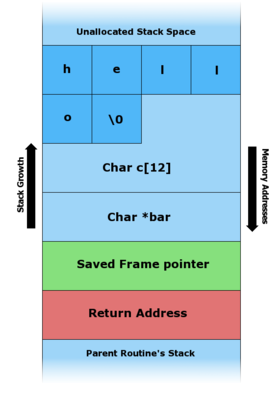
Здесь strcpy заменена на strncpy, в которой максимальное число копируемых символов ограничено размером буфера.

На схемах ниже видно, как уязвимая программа может повредить структуру стека.

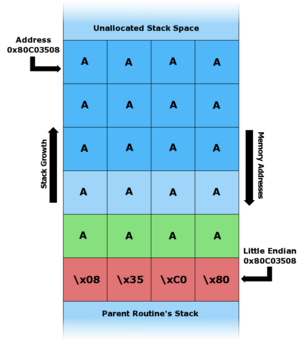
Иллюстрация записи различных данных в буфер, выделенный в стеке.



Перед копированием данных.



Строка «hello» была записана на стек



Буфер переполнен, что привело к перезаписи адреса возврата (return address).

В архитектуре x86 стек растёт от больших адресов к меньшим, то есть новые данные помещаются перед теми, которые уже находятся в стеке.

Записывая данные в буфер, можно осуществить запись за его границами и изменить находящиеся там данные, в частности, изменить адрес возврата.

Если программа имеет особые привилегии (например, запущена с правами root), злоумышленник может заменить адрес возврата на адрес шелл-кода, что позволит ему исполнять команды в атакуемой системе с повышенными привилегиями.

Техники применения переполнения буфера меняются в зависимости от архитектуры, операционной системы и области памяти. Например, случай с переполнением буфера в куче (используемой для динамического выделения памяти) значительно отличается от аналогичного в стеке вызовов.

**Эксплуатация в стеке**

Также известно, как Stack smashing. Технически подкованный пользователь может использовать переполнение буфера в стеке, чтобы управлять программой в своих целях, следующими способами:

* перезаписывая локальную переменную, находящуюся в памяти рядом с буфером, изменяя поведение программы в свою пользу.
* перезаписывая адрес возврата в стековом кадре. Как только функция завершается, управление передаётся по указанному атакующим адресу, обычно в область памяти, к изменению которой он имел доступ.
* перезаписывая указатель на функцию или обработчик исключений, которые впоследствии получат управление.
* перезаписывая параметр из другого стекового кадра или нелокальный адрес, на который указывается в текущем контексте.

Если адрес пользовательских данных неизвестен, но он хранится в регистре, можно применить метод «trampolining» (с англ. — «прыжки на батуте»): адрес возврата может быть перезаписан адресом опкода, который передаст управление в область памяти с пользовательскими данными. Если адрес хранится в регистре R, то переход к команде, передающей управление по этому адресу (например, call R), вызовет исполнение заданного пользователем кода. Адреса подходящих опкодов или байтов памяти могут быть найдены в DLL или в самом исполняемом файле. Однако адреса обычно не могут содержать нулевых символов, а местонахождения этих опкодов меняются в зависимости от приложения и операционной системы. Metasploit Project, например, хранил базу данных подходящих опкодов для систем Windows (на данный момент она недоступна).

Переполнение буфера в стеке не нужно путать с переполнением стека.

Также стоит отметить, что такие уязвимости обычно находят с помощью техники fuzz testing.

**Эксплуатация в куче**

Переполнение буфера в области данных кучи называется переполнением кучи и эксплуатируется иным способом, чем переполнение буфера в стеке. Память в куче выделяется приложением динамически во время выполнения и обычно содержит программные данные. Эксплуатация производится путём порчи этих данных особыми способами, чтобы заставить приложение перезаписать внутренние структуры, такие как указатели в связных списках. Обычная техника эксплойта для переполнения буфера кучи — перезапись ссылок динамической памяти (например, метаданных функции malloc) и использование полученного изменённого указателя для перезаписи указателя на функцию программы.

Уязвимость в продукте GDI+ компании Microsoft, возникающая при обработке изображений формата JPEG — пример опасности, которую может представлять переполнение буфера в куче.