

Искусство взлома и защиты систем

Дж. Козиол, Д. Личфилд, Д. Эйтэл, К. Энли и др.

В книге рассмотрены различные типы программного обеспечения: операционные системы, базы данных, интернет-серверы и т. д. На множестве примеров показано, как именно находить уязвимости в программном обеспечении. Тема особенно актуальна, так как в настоящее время в компьютерной индустрии безопасности программного обеспечения уделяется все больше внимания.

Содержание

Об авторах

От издательства

Благодарности

Часть I. Эксплуатация уязвимостей Linux на процессорах x86

Глава 1. Введение в эксплуатацию уязвимостей

Основные концепции

Управление памятью

Ассемблер

Программные конструкции C++ в ассемблере

Итоги

Глава 2. Переполнение стека

Буферы

Стек

Функции и стек

Переполнение буферов в стеке

Управление значением EIP

Использование уязвимостей для получения root-привилегий

Проблема адресации

Метод NOP

Борьба с неисполняемым стеком

Возврат в libc

Итоги

Глава 3. Внедряемый код

Системные функции

Написание внедряемого кода для вызова exit()

Устранение нуль-символов

Запуск командного процессора

Итоги

Глава 4. Дефекты форматных строк

Понятие форматной строки

Понятие дефекта форматной строки

Использование дефектов форматных строк

Аварийное завершение служб

Утечка информации

Контроль над исполнением

Причины дефектов форматных строк

Обзор приемов эксплуатации дефектов форматных строк

Итоги

Глава 5. Переполнение кучи

Понятие кучи

Функционирование кучи

Переполнение кучи

Основные принципы переполнения кучи

Объекты замены

Итоги

Часть II. Платформа Windows

Глава 6. Дикий мир Windows

Различие Windows и Linux

API и PE-COFF для Win32

Куча

Многопоточность

Гениальность и глупость концепций DCOM и DCE-RPC

Сбор информации

Эксплуатация уязвимостей

Маркеры и заимствование прав

Обработка исключений в Win32

Отладчики для Windows

Дефекты Win32

Написание внедряемого кода для Windows

Win32 API для хакера

Семейство Windows с точки зрения хакера

Итоги

Глава 7. Внедряемый код для Windows

Синтаксис и фильтры

Подготовка

Анализ блока PEB

Файл Heapoverflow.c

Поиск с использованием механизма обработки исключений Windows

Запуск командного процессора

Почему этого делать не следует

Итоги

Глава 8. Переполнение в Windows

Переполнение буфера в стеке

Стековые обработчики исключений
Манипуляции стековыми обработчиками исключений в Windows 2003 Server
Последнее замечание о перезаписи стековых обработчиков
Защита стека и Windows 2003 Server
Переполнение буфера в куче
Куча процесса
Динамическая куча
Работа с кучей
Функционирование кучи
Эксплуатация уязвимостей при переполнении кучи
Перезапись указателя на RtlEnterCriticalSection в PEВ
Перезапись указателя на первый векторный обработчик по адресу 77FC3210
Перезапись указателя на фильтр необработанных исключений
Перезапись указателя на обработчик исключения в блоке ТЕВ
Восстановление кучи
Другие аспекты переполнения кучи
Несколько слов в завершение
Другие варианты переполнения
Переполнение секции .data
Переполнение блоков ТЕВ и РЕВ
Переполнение буфера и неисполняемые стеки
Итоги

Глава 9. Обход фильтров

Код обхода алфавитно-цифровых фильтров
Код обхода Unicode-фильтров
Кодировка Unicode
Преобразование из ASCII в Unicode
Эксплуатация уязвимостей кодировки Unicode
Допустимый набор команд
Венецианский метод
ASCII-реализация венецианского метода
Дешифрирование
Код дешифрирования
Изменение адреса буфера
Итоги

Часть III. Выявление уязвимостей

Глава 10. Формирование рабочей среды

Справочные материалы
Средства разработки
gсс
gdb

NASM
WinDbg
OllyDbg
SoftICE
Visual C++
Python
Средства анализа
Полезные сценарии и утилиты
Все платформы
Unix
Windows
Базовые сведения
Архивы
Оптимизация процесса разработки внедряемого кода
Планирование
Встроенный ассемблер
Библиотеки встроенного кода
Корректное продолжение
Стабильность
Перехват подключения
Итоги

Глава 11. Внесение ошибок
Архитектура
Генератор входных данных
Внесение ошибок
Механизм модификации
Передача ошибок
Алгоритм Нейгла
Временные характеристики
Эвристика
Протоколы с поддержкой и без поддержки состояния
Мониторинг ошибок
Отладчик
Программа FaultMon
Все вместе
Итоги

Глава 12. Искусство фаззинга
Общая теория фаззинга
Статический анализ и фаззинг
Масштабируемость фаззинга
Недостатки фаззеров

Моделирование произвольных сетевых протоколов
Другие возможности фаззинга
Переключение битов
Модификация программ с открытыми исходными текстами
Фаззинг с динамическим анализом
Программа SPIKE
Копилка
Моделирование сетевых протоколов с помощью структуры данных SPIKE
Фаззеры SPIKE
Пример использования фаззера SPIKE
Другие фаззеры
Итоги

Глава 13. Анализ исходных текстов на языках семейства C

Инструменты
Cscope
Ctags
Редакторы
Cbrowser
Средства автоматического анализа исходных текстов
Методология
Нисходящий анализ
Восходящий анализ
Избирательный анализ
Классы уязвимостей
Общие логические ошибки
Пережитки прошлого
Форматные строки
Общие ошибки проверки границ
Циклические конструкции
Уязвимости единичного смещения
Ошибки некорректного завершения строк
Пропуск завершителя
Уязвимости знакового сравнения
Целочисленное переполнение
Преобразование целых чисел с разной разрядностью
Повторное освобождение памяти
Использование памяти вне области видимости
Использование неинициализированных переменных
Использование памяти после освобождения
Проблемы многопоточности и реентерабельности
Дефекты и реальные уязвимости

Итоги

Глава 14. Инструментальный анализ

Философия

Переполнение процесса extproc в Oracle

Типичные архитектурные дефекты

Пограничные проблемы

Проблемы преобразования данных

Проблемы мест дисбаланса

Проблемы различия между аутентификацией и авторизацией

Проблемы в самых очевидных местах

Обход процедур проверки входных данных и обнаружения атак

Удаление недопустимых данных

Использование альтернативных кодировок

Файловые операции

Обход сигнатур обнаружения атак

Борьба с ограничениями длины

Дефект SNMP-демона DOS в Windows 2000

Обнаружение DOS-атак

Дефект SQL-UDP

Итоги

Глава 15. Трассировка уязвимостей

Общие сведения

Уязвимая программа

Основные компоненты

Внедрение в адресное пространство процесса

Анализ машинного кода

Перехватчики

Сбор данных

Разработка VulnTrace

VTInject

Программа VulnTrace

Дополнительные приемы трассировки

Сигнатурный поиск

Другие классы уязвимостей

Итоги

Глава 16. Двоичный анализ

Очевидные различия двух видов анализа

IDA Pro — лучший инструмент

Краткий обзор возможностей IDA Pro

Отладочные символические имена

Введение в двоичный анализ

Стековый кадр
Конвенции вызова
Компиляторные конструкции
Аналоги функции memcpy
Аналоги функции strlen
Конструкции C++
Указатель this
Реконструкция определений классов
Таблицы виртуальных функций
Полезные факты
Ручные методы двоичного анализа
Просмотр библиотечных вызовов
Подозрительные циклы и команды записи
Анализ на более высоком уровне и логические ошибки
Графический анализ двоичных файлов
Ручная декомпиляция
Примеры двоичных уязвимостей
Дефекты Microsoft SQL Server
Уязвимость LSD RPC-DCOM
Уязвимость IIS WebDAV
Итоги

Часть IV. Дополнительные материалы

Глава 17. Альтернативные стратегии

Модификация программы
3-байтовая модификация SQL Server
1-битовая модификация MySQL
Аутентификация OpenSSH RSA
Другие стратегии модификации кода на стадии выполнения
Модификация GPG 1.2.2
Загрузка и запуск (проглет-сервер)
Опосредованный вызов системных функций
Проблемы опосредованного вызова
Итоги

Глава 18. Реальные условия, реальные проблемы

Факторы ненадежности
Волшебные числа
Версии
Проблемы с внедряемым кодом
Контрмеры
Подготовка
Метод “грубой силы”

Локальные решения
Идентификация ОС и приложения
Утечки информации
Итоги

Глава 19. Атаки на СУБД

Атаки сетевого уровня
Атаки прикладного уровня
Выполнение команд операционной системы
Microsoft SQL Server
Oracle
IBM DB2
Атаки на уровне SQL
SQL-функции
Итоги

Глава 20. Переполнение в ядре

Типы уязвимостей ядра
Переполение буфера в стеке ядра OpenBSD
Перезапись памяти ядра в OpenBSD
Утечка информации из памяти ядра в FreeBSD
Уязвимость `pricntl()` в Solaris
Новые уязвимости ядра
Переполение в функции `exec_ibcs2_coff_prep_zmagic()` ядра OpenBSD
Уязвимость перебора загружаемых модулей ядра `vfs_getvfssw()` в Solaris
Итоги

Глава 21. Эксплуатация уязвимостей ядра

Уязвимость `exec_ibcs2_coff_prep_zmagic()`
Вычисление смещений и контрольных точек
Замена адреса возврата и перехват управления
Получение дескриптора процесса
Создание внедряемого кода режима ядра
Возврат из режима ядра
Получение root-привилегий (`uid=0`)
Уязвимость загрузки модулей ядра `vfs_getvfssw()`
Разработка кода
Загружаемый модуль ядра
Получение root-привилегий (`uid=0`)
Итоги
Алфавитный указатель

Об авторах

Джек Козиол, основной автор книги, работает старшим преподавателем и руководителем программ компьютерной безопасности в институте InfoSec. Его регулярно приглашают для подготовки персонала в разведке, вооруженных силах и органах правопорядка США. Кроме того,

Джек проводит обучение во многих крупнейших компаниях, включая Microsoft, HP и Citibank, в области защиты сетей и приложений. В свободное время от проведения учебных семинаров Джек занимается испытаниями на проникновение в системы и анализом безопасности приложений для многочисленных клиентов. Он обладает многолетним опытом выявления и эксплуатации уязвимостей — как по заказам клиентов, так и по собственной инициативе.

Козиол также является автором “Intrusion Detection with Snort”, одной из самых популярных книг в области безопасности (2003). Книга была переведена на несколько языков, включая французский и японский, и получила высокие оценки в журналах Linux Journal, Slashdot и Information Security. Джек выступал в USA Today, CNN, MSNBC, First Business и других средствах массовой информации по вопросам информационной безопасности. Он живет в Оук-Парк, штат Иллинойс, неподалеку от дома и студии Фрэнка Ллойда Райта, со своей подружкой Трейси и собакой Квази.

Дэвид Личфилд, один из ведущих мировых специалистов в области компьютерной безопасности, является одним из пяти основателей NGSSoftware. Дэвид обнаружил и опубликовал более 100 серьезных дефектов безопасности в разных продуктах, включая Apache, Microsoft Internet Information Server, Oracle и Microsoft SQL Server. Благодаря огромному опыту в области защиты и взлома сетей и приложений Дэвид постоянно ведет семинары на конференциях Black Hat Briefings. Также является основным автором книги “SQL Security” (издательство Osborne/McGraw-Hill).

Дэйв Аител — автор пакета SPIKE и основатель компании “Immunity, Inc.”, занимающейся безопасностью в Интернете. Его работа связана с эксплуатацией уязвимостей на платформах Windows и Unix и разработкой методологий поиска новых уязвимостей.

Крис Анли возглавляет Next Generation Security Software — расположенную в Великобритании компанию, консультациями, анализом и разработкой программного обеспечения в области безопасности. Крис активно занимается поиском уязвимостей; он является автором ряда официальных документов и статей по безопасности ряда продуктов, включая PGP, Windows, SQL Server и Oracle. Его свободное время равномерно распределяется между анализом, программированием и консультациями.

Синан Эрен, аналитик в области компьютерной безопасности, провел обширную работу по эксплуатации уязвимостей Unix и разработке нетривиальных методологий использования дефектов уровня ядра; обнаружил множество серьезных дефектов в коммерческих и бесплатных программах Unix.

Нил Мехта занимается анализом безопасности приложений в ISS X-Force. Как и многие специалисты в области безопасности, раньше занимался инженерным анализом и реконструкцией кода. Получил богатый практический опыт, занимаясь консультациями в области защиты от копирования, но в последнее время его основным профилем деятельности стала безопасность приложений. Нил провел обширные исследования в области анализа двоичных файлов и исходных текстов, применяя полученные знания для выявления многочисленных уязвимостей в широко распространенных сетевых приложениях.

Райли Хассел, старший инженер в eEye Digital Security, отвечает за разработку и реализацию систем контроля качества и аналитического инструментария. Также обнаружил ряд известных уязвимостей, опубликованных eEye Digital Security.

Глава 18. Реальные условия, реальные проблемы

У каждого дефекта есть своя история. Дефекты рождаются, живут и умирают, нередко так и оставаясь скрытыми и неиспользованными. Для хакера любой дефект открывает прекрасный случай создать код «магического заклинания», которое превращает уязвимую стену в дверь. Но одно дело — создать решение для лабораторной среды и совсем другое — для электронных джунглей современного Интернета. Эта глава посвящена решениям для реальных условий.

Факторы ненадежности

фактор ненадежности В этой части главы рассматриваются различные причины, из-за которых ваше решение может не работать в реальных условиях. Впрочем, вас не должно пугать обилие возможных проблем — как говорится, «даже слепая курица иногда находит зерно».

Волшебные числа

волшебное число Некоторые уязвимости (такие, как переполнение стека RealServer, описанное в главе 12) поддаются надежной эксплуатации. Другие (скажем, дефект двойного вызова функции free() в dtlogin) практически невозможно эксплуатировать сколько-нибудь стабильно. Тем не менее,

судить о надежности решения можно только после того, как вы попытаетесь применить его на практике.

Кроме того, эксплуатация все более и более сложных уязвимостей является единственным способом освоения новых методов. Сколько бы вы ни читали о том или ином приеме, вы никогда не научитесь реально его применять. Исходя из этого, следует всегда прикладывать дополнительные усилия к тому, чтобы ваши решения работали как можно надежнее. Иногда решение стабильно работает в лаборатории, но только на 50 % срабатывает в реальных условиях; тогда, чтобы повысить надежность, его часто приходится переписывать заново.

Может оказаться, что первая версия кода, написанного для конкретной уязвимости, работает только на вашем компьютере. Обычно это говорит о том, что в коде были жестко закодированы некоторые важные значения (скорее всего, адрес возврата или адрес `geteip`). Когда представляется возможность заменить указатель на функцию или сохраненный адрес возврата, управление надо куда-то передать. Конкретный адрес передачи может зависеть от многих факторов, лишь часть из которых находится под вашим контролем. В контексте данной главы мы назовем подобную ситуацию *однофакторным эксплойтом* (однофакторный (one-factor exploit)).

Также в коде может иметься место, по которому находится указатель на строку. Целевая программа использует этот указатель перед тем, как вы берете ее под свой контроль. Для предотвращения сбоев этот указатель (часть атакующей строки) необходимо установить на некоторое безопасное место в памяти. Этот шаг создает второй фактор, необходимый для успешной эксплуатации уязвимости.

Большинство простых эксплойтов являются одно- и двухфакторными эксплойтами. Например, эксплойт, реализующий простейшее удаленное переполнение стека, обычно является однофакторным — вы должны лишь правильно задать адрес внедряемого кода в памяти. Но с переходом к эксплойтам, основанным на переполнении кучи, которые обычно являются двухфакторными, приходится искать пути к снижению степени хаоса в системе.

Версии

При эксплуатации уязвимостей в реальных условиях возникает одна важная проблема: вы далеко не всегда знаете, какие программы работают на атакуемом компьютере. Возможно, это окажется компьютер с Windows 2000 Advanced Server, как в вашей лаборатории; а может быть, на нем работает ColdFusion и происходят постоянные перемещения в памяти. Нельзя исключать и того, что на нем установлена версия Windows 2000 для упрощенного китайского письма. На компьютере могут быть установлены любые обновления вплоть до последней версии Service Pack; может оказаться, что некоторые из них устанавливались вручную, причем, возможно, некорректно. С другой стороны, удаленный хост может работать под управлением Linux для платформы Alpha или иметь SMP-архитектуру. Многие опубликованные уязвимости Microsoft RPC Locator не работают на компьютерах SMP-архитектуры и в системах с процессором Xeon, которые Windows считает двухпроцессорными. Такие проблемы очень трудно решить в удаленном режиме.

Кроме того, при эксплуатации уязвимостей, основанных на повреждении кучи, возникают проблемы, не позволяющие другим пользоваться атакуемой службой. Еще одна распространенная проблема переполнения кучи заключается в том, что заменяемые указатели на функции зависят от конкретной версии `libc`. Так как в разных версиях Linux используются разные версии `libc`, это означает, что решение должно быть привязано к конкретному дистрибутиву Linux. К сожалению, ни один дистрибутив не завоевал доминирующего положения, поэтому жестко закодировать эти значения не так просто, как в случае с Windows или коммерческими версиями Unix.

Помните, что многие поставщики выпускают разные версии Unix с одинаковыми номерами версий. Ваша версия Solaris 8 CD может отличаться от другой версии Solaris 8 CD, если они приобретались в разное время. В вашей версии могут быть установлены заплатки, отсутствующие в другой версии, и наоборот.

Проблемы с внедряемым кодом

внедряемый код;внедряемый Некоторые программисты тратят целые недели на написание собственного внедряемого кода. Другие используют пакеты от Packetstorm (<http://packetstorm.org>). Но каким бы хитроумным ни был ваш внедряемый код, он все равно остается программой, написанной на ассемблере и выполняемой в нестабильной среде. Это означает, что причины неудач часто кроются в самом внедряемом коде.

В лаборатории атакующий и атакуемый компьютеры подключены к одному концентратору Ethernet. Но в реальных условиях цель может находиться на другом континенте под контролем другого

администратора, который настроил свою сеть по собственному вкусу. В частности, он может назначить размер MTU (Maximum Transmission Unit — максимальная единица передачимаксимальная единица передачи) равным 512, заблокировать ICMP, поддерживать фильтрацию на брандмауэре или создавать вам иные проблемы.

Проблемы с внедряемым кодом можно разделить на несколько категорий.

Сетевые проблемы

Размер MTU и маршрутизация могут стать серьезным источником проблем для внедряемого кода. Иногда вы атакуете один IP-адрес, а ответ приходит с другого IP-адреса или интерфейса. Еще одной распространенной проблемой является выходная фильтрация. Если переданный внедряемый код не может связаться с вами из-за фильтрации, он должен корректно завершить свою работу. Возможно, в него стоит включить код обратной связи через UDP- и ICMP-порт.

Проблемы привилегий

В Windows программный поток может не обладать привилегиями, необходимыми для загрузки `ws2_32.dll`. Обычное решение в таких ситуациях — перехватить поток, с которого производился вход, и считать, что библиотека `ws2_32.dll` уже загружена, или вызвать `RevertToSelf()`. Аналогичные проблемы с привилегиями возникают и в некоторых версиях Linux (SELinux и т. д.).

В отдельных редких случаях вам может быть запрещено подключаться к сокетам или прослушивать порты. В подобных ситуациях можно перехватить контроль над выполнением исходной программы (например, отключить нормальную аутентификацию целевого процесса, изменить файл, из которого осуществляется чтение, дополнить список пользователей и т. д.) или найти другой способ, при помощи которого внедряемый код сможет повысить свой уровень доступа без содействия извне.

Проблемы конфигураций

Неверная идентификация операционной системы может привести к тому, что внедряемый код окажется неправильным или будут использованы ошибочные адреса возврата. В удаленном режиме трудно отличить Alpha Linux от SPARC Linux; может быть, проще всего опробовать оба варианта.

Если целевой процесс находится под воздействием `chroot`, файл `/bin/sh` может и не существовать. Это еще одна веская причина, чтобы не использовать стандартный внедряемый код `exeve (/bin/sh)`.

Иногда база стека меняется в зависимости от типа процессора. Кроме того, не все команды действительны на всех типах процессоров. Например, может оказаться, что целевая система работает на старом чипе Alpha, а внедряемый код тестировался только на новом чипе. А может быть, новый атакуемый компьютер содержит большой кэш команд, который не был очищен во время атаки.

Проблемы IDS-хостов

Технологии `Chroot`, `LIDS`, `SELinux`, `BSD jail()`, `gsecurity`, `Papillion` и другие вариации на эти темы могут создать проблемы для внедряемого кода на многих уровнях. По мере того, как эти технологии становятся все более популярными, ожидайте, что вам придется иметь с ними дело во внедряемом коде. Узнать, помешают они вам или нет, можно только одним способом — установить их и попробовать. `Oken` и `Entercept` перехватывают вызовы системных функций и сравнивают их с вызовами, которые обычно используются в данном приложении. Возможны два решения: имитировать нормальное поведение приложения и стараться по возможности держаться в его рамках, или попытаться самостоятельно победить механизм перехвата. Если вы эксплуатируете уязвимость ядра, настало время делать это прямо из внедряемого кода.

Проблемы программных потоков

При переполнении кучи другой программный потокпрограммный потокпоток;программный может активизироваться для обработки запроса и попытаться вызвать `free()` или `malloc()`. Так как управляющие структуры кучи испорчены, это приведет к аварийному завершению процесса.

Другой поток может наблюдать за тем, чтобы ваш поток завершился в положенное время. Если поток перешел под ваш контроль, процесс-наблюдатель может «убить» его для восстановления нормальной работы. Попробуйте проверить наличие синхронизирующих потоков во внедряемом коде и имитировать их сигналы, если это возможно.

Может оказаться, что работа эксплойта зависит от адреса возврата, действительно только для одного потока; обычно это свидетельствует о недостаточном тестировании.

Контрмеры

Нестабильность приложения или его полный отказ может объясняться многими причинами. Тем не менее, существует много способов решить возникшие проблемы. Важно помнить, что вы пишете приложение, которого по идее существовать не должно; оно появляется на свет только из-за дефектов в других программах. В процессе работы над внедряемым кодом следует постоянно искать альтернативные средства решения возникающих проблем. Если вы не уверены, задайте себе вопрос: «А что бы на моем месте сделал Джон МакДональд?» Далее приводится выдержка из журнала «Phrack» (номер 60, декабрь 2002 г.), характеризующая его философию. Помните его слова всегда, когда у вас возникнут проблемы.

Корреспондент: В прошлом вы нашли немало дефектов в программах и написали код для их эксплуатации. Некоторые уязвимости требовали новых нестандартных концепций, неизвестных в то время. Что заставляет вас заниматься эксплуатацией сложных дефектов и какие методы вы использовали?

Джон МакДональд: Мои мотивы изменялись со временем. Я мог бы привести несколько второстепенных причин, которые управляли моими поступками на разных этапах моей жизни; среди них были как эгоистические, так альтруистические. Но мне кажется, что на самом деле все сводится к желанию докопаться до сути происходящего. Что касается методов, я стараюсь действовать систематично. Я выделяю большую часть времени на чтение программы, пытаюсь разобраться в ее архитектуре, понять подход автора и используемые им приемы. Это также помогает настроить мое подсознание в нужном направлении.

Я обычно начинаю с нижних уровней программы или системы и поиска любых потенциально неожиданных аспектов поведения, которые могут переходить на верхние уровни. Я документирую каждую функцию и анализирую все потенциальные проблемы, которые в ней есть. Время от времени я отрываюсь от документации и занимаюсь куда более интересным делом, проверяя свои теории и выясняя возможность их практического применения.

В том, что касается написания кода, я обычно стараюсь свести к минимуму или полностью ликвидировать все факторы, в которых требуется что-то угадывать.

Если решение почти готово, но в какой-то момент дело застопоривается, попробуйте взглянуть на происходящее с новой точки зрения. Работайте «в стиле Алвара» — тратьте много времени в IDA Pro, подробно изучая точное местонахождение каждой ошибки и все, что программа делает после нее. Бомбардируйте программу сверхдлинными строками. Проанализируйте, что происходит в программе, когда она не «погибает» от вашей атаки. Возможно, удастся найти другой дефект, который будет работать надежнее.

Часто полезно ознакомиться с методами эксплуатации уязвимостей, используемыми на других платформах. Методы эксплуатации уязвимостей для Windows могут пригодиться в Unix, и наоборот. Но даже если они и не пригодятся, возможно, они дадут вам вдохновение, необходимое для разработки качественного эксплойта.

Подготовка

Будьте готовы к чему угодно. Всегда держите под рукой стопку жестких дисков со всеми ОС на всех языках со всеми доступными обновлениями Service Pack и заплатками; будьте готовы к перекрестному сравнению адресов — возможно, потребуются узнать, какие адреса работают на всех целевых компьютерах. VMWare окажет в этом неоценимую помощь, хотя иногда VMWare конфликтует с OllyDbg. Поиск по базе данных всех возможных адресов также сэкономит время по сравнению с методом «грубой силы».

Метод «грубой силы»

Иногда для устойчивости приложения проще всего перебрать все возможные «волшебные числа». Если у вас имеется большой список потенциальных адресов возврата по ebx, возможно, стоит перебрать их все. Так или иначе, метод «грубой силы» часто является последней мерой, но по крайней мере это абсолютно законная полезная мера.

Впрочем, существует ряд трюков, благодаря которым вы избежите от лишних затрат времени и не оставите ненужных следов в журналах атакуемой системы. Определите, можно ли проверять сразу несколько адресов методом «грубой силы». Кэшируйте все действительные результаты, чтобы позднее их можно было проверить в первую очередь. Компьютеры одной сети часто

настраиваются одинаковым образом, поэтому если ваша методика сработала один раз, вероятно, она сработает снова.

Иногда отправка непомерно больших буферов с внедряемым кодом дает разумные шансы правильно попасть на «волшебное число». И если возможно, попытайтесь выявить связи между «волшебными числами». Если вы знаете, что один необходимый адрес всегда находится рядом с другим, это значительно облегчит вашу работу по сравнению с поиском полностью независимых адресов.

Утечка памяти также часто упрощает метод «грубой силы». Иногда для заполнения памяти внедряемым кодом даже не требуется реальной утечки памяти. Например, в уязвимости IIS ColdFusion от CANVAS создается 1000 подключений к удаленному хосту, каждое из которых пересылает 20 000 байт внедряемого кода и NOP. Эта процедура быстро заполняет память копиями внедряемого кода. Затем без отключения остальных сокетов производится переполнение кучи. Местонахождение внедряемого кода необходимо угадывать, но догадка почти всегда оказывается правильной, потому что большая часть памяти процесса заполнена копиями кода.

Заполнение памяти процесса легко реализуется в случае многопоточных процессов, таких как IIS. Даже если процесс не является многопоточным, нужного результата часто удается добиться благодаря утечке памяти. А если найти утечку памяти не удастся, возможно, найдется статическая переменная, которая содержит последний результат запроса и находится по одному и тому же адресу. Если просмотреть всю программу в поисках операций, которыми можно манипулировать для достижения подобных целей, почти всегда найдется что-нибудь полезное.

Локальные решения

Ненадежность локального эксплойта ничем оправдать нельзя. Под вашим контролем находится практически все — пространство памяти, сигналы, содержимое диска и местонахождение текущего каталога. Многие разработчики при разработке локальных эксплойтов создают себе слишком много проблем. Если локальный эксплойт работает не в 100 % случаев, скорее всего, его создавал новичок.

Например, при организации простого переполнения локального буфера для платформ Linux/Unix используйте функцию `exeve()`, чтобы выяснить окружение целевого процесса. После этого вы сможете точно определить местонахождение внедряемого кода в памяти и реализовать атаку (скажем, возврата в `libc`) без каких-либо предположений. Лично мы предпочитаем выполнить возврат в `strcpy()`, скопировать внедряемый код в кучу и выполнить его там. Для определения адреса `strcpy()` используются функции `dlopen()` и `dlsym()`. Такие ухищрения повышают устойчивость ваших решений в условиях реальной эксплуатации.

Как указал наш соавтор Синан Эрен (известный в узких кругах как *noir*), при атаке ядра возможно отображение памяти на любой адрес, что позволяет установить адрес возврата непосредственно на начало внедряемого кода (другими словами, при реализации локальной атаки ядра адрес `0x00000000` может быть совершенно нормальным адресом возврата).

Идентификация ОС и приложения

Информация, получаемая программами `Nmap` или `Crackmap`, не всегда дает полную картину. При эксплуатации уязвимости приложения одной лишь версии операционной системы недостаточно. Также необходимо знать следующее:

- архитектура (x86/SPARC/другие);
- версия приложения;
- конфигурация приложения;
- конфигурация операционной системы (неисполняемый стек/PaX/др.).

Нередко идентификация операционной системы оказывается абсолютно бесполезной, потому что происходит перенаправление с одного хоста на другой. А может быть, вам просто нельзя передавать пакеты идентификации ОС на хост, чтобы не привлекать внимания систем IDS, прослушивающих сеть. Таким образом, для написания надежного решения часто приходится искать уникальные способы идентификации удаленного хоста, не выходящие за рамки обычного трафика.

Всегда лучше всего проводить идентификацию на том же порте, через который будет проводиться атака. Рассмотрим реальный пример, задействованный в эксплойте CANVAS MSRPC. Простое использование порта 135 (целевой службы) позволяет получить информацию об операционной системе. Сначала мы отделяем XP и Windows 2003 от NT 4.0 и Windows 2003. Затем 2003

отделяется от XP (эта полезная функция здесь не показана), а Windows 2000 отделяется от NT 4.0. Функция использует общедоступные интерфейсы на порте 135 (TCP); конечно, это хорошо, потому что другие открытые порты могут отсутствовать. При выборе этой методики удается определить платформу всего несколькими подключениями.

```
def runTest(self):
    UUID2K3="1d55b526-c137-46c5-ab79-638f2a68e869"
    callid=1
    error,s=msrpcbind(UUID2K3,1,0,self.host,self.port,callid)
    if error==0:
        errstr="Could not bind to the msrpc service for 2K3,XP -
        assuming NT 4 or Win2K"
        self.log(errstr)
    else:
        if self.testFor2003(): # Простая проверка здесь не приводится
            self.setVersion(15)
            self.log(
                "Test indicated connection succeeded to msrpc service.")
            self.log("Attacking using version %d:
            %s"%(self.version,self.versions[self.version][0]))
            return 1
        self.setVersion(1) # По умолчанию Win2K или XP
        UUID2K="000001a0-0000-0000-c000-000000000046"
        # Поддерживается в 2K и выше
        callid=1
        error,s=msrpcbind(UUID2K,0,0,self.host,self.port,callid)
        if error==0:
            errstr="Could not bind to the msrpc service for 2K and above -
            assuming NT 4"
            self.log(errstr)
            self.setVersion(14) #NT4
        else:
            self.log("Test indicated connection succeeded to msrpc
            service.")
            self.log("Attacking using version %d:
            %s"%(self.version,self.versions[self.version][0]))
            return 1 #Windows 2000 èèèè XP
        callid=0
        #IRemoteDispatch UUID
        UUID="4d9f4ab8-7d1c-11cf-861e-0020af6e7c57"
        error,s=msrpcbind(UUID,0,0,self.host,self.port,callid)
        if error==0:
            errstr="Could not bind to the msrpc service necessary
            to run the attack"
            self.log(err.str)
            return 0
        # Если проверка прошла успешно, считаем, что что служба уязвима
        self.log("Test indicated connection succeeded to msrpc service.")
        self.log("Attacking using version %d:
        %s"%(self.version,self.versions[self.version][0]))
        return 1
```

Утечки информации

Прошли те времена, когда атаки напоминали стрельбу неуправляемыми ракетами. В наши дни хороший разработчик ищет способы направить свою атаку точно в цель. Существуют различные методы получения информации от цели атаки (нередко с конкретных адресов памяти). Далее перечислены некоторые из них.

- Чтение и интерпретация данных, передаваемых объектом атаки. Например, MSRPC-пакеты часто содержат указатели, путем продвижения продвижение (marshalling) передаваемые прямо из памяти. По их значениям можно составить представление о структуре памяти целевого процесса.
- Переполнение кучи для записи данных перед отправкой позволяет узнать, где именно в памяти находится буфер.
- Переполнение кучи атака;переполнения кучи в стиле frontlink() для записи адреса внутренних переменных malloc в данные перед отправкой при помощи несложных вычислений позволяет определить, где находятся указатели функции malloc.

- Перезапись поля длины атака; перезаписи поля длины часто позволяют получить содержимое больших блоков серверной памяти (см. переполнение BIND TSIG).
- Опустошения (underflow) и другие аналогичные атаки также могут использоваться для приема содержимого блоков памяти сервера. Аналитик FX из группы Phenoelit успешно применял этот метод с эхо-пакетами при эксплуатации уязвимости Cisco HTTPD. Его работа является блестящим примером объединения двух решений для получения одного очень надежного решения.

Анализ временных меток; временных меток также может дать ценную информацию об ошибках, происходящих при работе вашего кода. Был ли отправлен пакет сброса немедленно или только по прошествии тайм-аута?

Алвар Флейк однажды сказал: «Хороший хакер не ограничивается одним дефектом». Утечка информации может открыть доступ к эксплуатации даже очень «трудных» дефектов.

Итоги

Допустим, вы пишете код атаки на веб-сервер Win32. По прошествии суток ваш эксплойт, реализующий простое переполнение стека, прекрасно работает пять раз из шести. В нем используется стандартная методика «замены структуры обработчика исключения». В свою очередь, это приводит к выполнению последовательности команд pop pop ret в сегменте .text. Но поскольку целевой процесс является многопоточным, иногда внедряемый код перезаписывается другим потоком, и атака завершается неудачей. Следовательно, код стоит переписать с гораздо меньшей строкой, обеспечивающей безопасный возврат из функции, и со временем получить контроль через сохраненный указатель возврата, находящийся на расстоянии в несколько кадров стека. Хотя данная методика ограничивает объем внедряемого кода, она работает гораздо надежнее.

Этим примером мы хотели сказать, что в некоторых случаях нельзя полностью полагаться даже на очень надежные методы — иногда приходится тестировать несколько разных методов эксплуатации уязвимости, а затем опробовать каждый метод на как можно большем количестве тестовых платформ, пока не будет найдено оптимальное решение. Если работа заходит в тупик, попробуйте сделать строку атаки как можно длиннее или как можно короче; внедрите в нее символы, присутствие которых изменяет режим работы эксплойта. Если доступен исходный текст, попробуйте тщательно проанализировать перемещения данных в программе. Главное — не падайте духом. Работа в области безопасности требует большого терпения; ведь в успехе можно быть уверенным не ранее того момента, когда эксплойт наконец заработает.

Уверяем, настойчивость окупится. Но в отдельных случаях вы никогда не узнаете, почему же ваш эксплойт так и не заработал в реальных условиях, и с этим придется смириться.