Руководство FreeBSD для разработчиков

Аннотация

Добро пожаловать в руководство FreeBSD для разработчиков.

Содержание

Часть I: Введение

Глава 1. Разработка во FreeBSD

Здесь необходимо будет обсудить FreeBSD в качестве платформы для разработки, подход к этому BSD, обзор архитектуры, структура /usr/src, история и так далее.

Спасибо вам за выбор FreeBSD в качестве платформы разработки! Надеемся, что она вас не подведет.

Глава 2. Парадигма BSD

Глава 3. Обзор архитектуры

Глава 4. Структура /usr/src

Полный исходный код FreeBSD имеется на нашем общедоступном хранилище CVS. Как правило, исходный код устанавливается в каталог /usr/src, содержащий следующие подкаталоги.

Каталог	**Описание**
bin/	Исходный код файлов из /bin
contrib/	Исходный код файлов программного обеспечения, которое предоставлено третьими лицами.
crypto/	Исходный код DES
etc/	Исходный код файлов из /etc
games/	Исходный код файлов из /usr/games
gnu/	Утилиты, подпадающие под условия GNU Public License
include/	Исходный код файлов из /usr/include
kerberosIV/	Исходный код Kerbereros версии IV
kerberos5/	Исходный код Kerbereros версии 5
lib/	Исходный код файлов из /usr/lib
libexec/	Исходный код файлов из /usr/libexec
release/	Файлы, необходимые для выпуска релиза FreeBSD
sbin/	Исходный код файлов из /sbin
secure/	Исходный код FreeSec
share/	Исходный код файлов из /sbin
sys/	Исходные тексты ядра
tools/	Инструментальные средства, используемые для обслуживания и тестирования FreeBSD
usr.bin/	Исходный код файлов из /usr/bin
usr.sbin/	Исходный код файлов из /usr/sbin

Часть II: Основы

Глава 5. Безопасное программирование

5.1. Обзор

Эта глава описывает некоторые из проблем обеспечения безопасности, которые десятилетиями преследовали программистов UNIX®, а также несколько новых доступных инструментов, помогающих программистам избежать написания небезопасного кода.

5.2. Методология обеспечения безопасности

Написание безопасных приложений требует весьма критического и пессимистического взгляда на жизнь. Приложения должны работать по принципу "наименьших привилегий", при котором никакой процесс не должен работать с привилегиями, превышающими минимально необходимый для выполнения своих функций минимум. Ранее проверенный код должен использоваться там, где только это возможно для избежания общих ошибок, которые могли быть уже исправлены другими.

Одной из неприятностей в среде UNIX® является легкость в предположении безопасности этого окружения. Приложения никогда не должны верить пользовательскому вводу (во всех его формах), ресурсам системы, межпроцессному взаимодействию или времени выполнения событий. Процессы UNIX® выполняются не синхронно, так что логические операции редко бывают атомарными.

5.3. Переполнения буфера

Переполнения буфера появились вместе с появление архитектуры Фон-Неймана Впервые широкую известность они получили в 1988 году вместе с Интернет-червем Морриса (Morris). К сожалению, точно такая же атака остаётся эффективной и в наши дни. Из 17 бюллетеней безопасности CERT за 1999 год, 10 были непосредственно вызваны ошибкам в программном обеспечении, связанным с переполнениями буфера. Самые распространенные типы атак с использованием переполнения буфера основаны на разрушении стека.

Самые современные вычислительные системы используют стек для передачи аргументов процедурам и сохранения локальных переменных. Стек является буфером типа LIFO (последним вошел первым вышел) в верхней части области памяти процесса. Когда программа вызывает функцию, создается новая "граница стека". Эта граница состоит из аргументов, переданных в функцию, а также динамического количества пространства локальных переменных. "Указатель стека" является регистром, хранящим текущее положение вершины стека. Так как это значение постоянно меняется вместе с помещением новых значений на вершину стека, многие реализации также предусматривают "указатель границы", который расположен около начала стека, так что локальные переменные можно легко адресовать относительно этого значения. Адрес возврата из функции также сохраняется в стеке, и это является причиной нарушений безопасности, связанных с переполнением стека, так как перезаписывание локальной переменной в функции может изменить адрес возврата из этой функции, потенциально позволяя злоумышленнику

выполнить любой код.

Хотя атаки с переполнением стека являются самыми распространенными, стек можно также перезаписать при помощи атаки, основанной на выделении памяти (malloc/free) из "кучи".

Как и во многих других языках программирования, в C не выполняется автоматической проверки границ в массивах или указателях. Кроме того, стандартная библиотека C полна очень опасных функций.

5.4. Пример переполнения буфера

В следующем примере кода имеется ошибка переполнения буфера, предназначенная для перезаписи адреса возврата и обхода инструкции, следующей непосредственно за вызовом функции. (По мотивам)

```
#include stdio.h

void manipulate(char *buffer) {
  char newbuffer[80];
  strcpy(newbuffer,buffer);
}

int main() {
  char ch,buffer[4096];
  int i=0;

  while ((buffer[i++] = getchar()) != '\n') {};

  i=1;
  manipulate(buffer);
  i=2;
  printf("The value of i is : %d\n",i);
  return 0;
}
```

Давайте посмотрим, как будет выглядеть образ процесса, если в нашу маленькую программу мы введем 160 пробелов.

Очевидно, что для выполнения реальных инструкций (таких, как exec(/bin/sh)), может быть придуман более вредоносный ввод.

5.4.1. Как избежать переполнений буфера

Самым прямолинейным решением проблемы переполнения стека является использование только памяти фиксированного размера и функций копирования строк. Функции `strncpy` и `strncat` являются частью стандартной библиотеки C. Эти функции будут копировать не более указанного количества байт из исходной строки в целевую. Однако у этих функций

есть несколько проблем. Ни одна из них не гарантирует наличие символа NUL, если размер входного буфера больше, чем целевого. Параметр длины также по-разному используется в strncpy и strncat, так что для программистов легко запутаться в правильном использовании. Есть также и значительная потеря производительности по сравнению с `strcpy` при копировании короткой строки в большой буфер, потому что `strncpy` заполняет символами NUL пространство до указанной длины.

Для избежания этих проблем в OpenBSD была сделана другая реализация копирования памяти. Функции `strlcpy` и `strlcat` гарантируют, что они они всегда терминируют целевую строку нулевым символом, если им будет передан аргумент ненулевой длины. Более подробная информация об этом находится здесь Инструкции OpenBSD `strlcpy` и `strlcat` существуют во FreeBSD начиная с версии 3.3.

5.4.1.1. Вкомпилированная проверка границ во время выполнения

К сожалению, все еще широко используется очень большой объём кода, который слепо копирует память без использования только что рассмотренных функций с проверкой границ. Однако есть другое решение. Существует несколько расширений к компилятору и библиотек для выполнения контроля границ во время выполнения (C/C++).

Одним из таких добавлений является StackGuard, который реализован как маленький патч к генератору кода gcc. Согласно web сайту StackGuard:

"StackGuard распознает и защищает стек от атак, не позволяя изменять адрес возврата в стеке. При вызове функции StackGuard помещает вслед за адресом возврата сигнальное слово. Если после возврата из функции оно оказывается измененным, то была попытка выполнить атаку на стек, и программа отвечает на это генерацией сообщения о злоумышленнике в системном журнале, а затем прекращает работу."

"StackGuard реализован в виде маленького патча к генератору кода gcc, а именно процедур function_prolog() и function_epilog(). function_prolog() усовершенствована для создания пометок в стеке при начале работы функции, а function_epilog() проверяет целостность пометки при возврате из функции. Таким образом, любые попытки изменения адреса возврата определяются до возврата из функции."

Перекомпиляция вашего приложения со StackGuard является эффективным способом остановить большинство атак переполнений буфера, но все же полностью это проблемы не решает.

5.4.1.2. Проверка границ во время выполнения с использованием библиотек.

Механизмы на основе компилятора полностью бесполезны для программного обеспечения, поставляемого в двоичном виде, которое вы не можете перекомпилировать. В этих ситуациях имеется некоторое количество библиотек, в которых реализованы небезопасные функции библиотеки C (`strcpy`, `fscanf`, `getwd`, и так далее..), обеспечивающие невозможность записи после указателя стека.

- libsafe
- libverify

- libparanoia

К сожалению, эти защиты имеют некоторое количество недостатков. Эти библиотеки могут защитить только против малого количества проблем, и не могут исправить реальные проблемы. Эти защиты могут не сработать, если приложение скомпилировано с параметром -fomit-frame-pointer. К тому же переменные окружения LD_PRELOAD и LD_LIBRARY_PATH могут быть переопределены/сняты пользователем.

5.5. Проблемы с установленным битом UID

Имеется по крайней мере 6 различных идентификаторов (ID), связанных с любым взятым процессом. Поэтому вы должны быть очень осторожны с тем, какие права имеет ваш процесс в каждый момент времени. В частности, все seteuid-приложения должны понижать свои привилегии, как только в них отпадает необходимость.

Реальный ID пользователя может быть изменен только процессом администратора. Программа login устанавливает его, когда пользователь входит в систему, и он редко меняется.

Эффективный ID пользователя устанавливается функциями `exec()`, если у программы установлен бит seteuidt. Приложение может выполнить вызов `seteuid()` в любой момент для установки эффективного ID пользователя в значение реального ID пользователя или сохраняемого set-user-ID. Когда эффективный ID пользователя устанавливается функциями `exec()`, его предыдущее значение сохраняется в сохраняемом set-user-ID.

5.6. Ограничение среды работы вашей программы

Традиционно используемым методом ограничения процесса является использование системного вызова `chroot()`. Этот системный вызов меняет корневой каталог, относительно которого определяются все остальные пути в самом процессе и всех порожденных ими процессах. Для того, чтобы этот вызов был выполнен успешно, процесс должен иметь право на выполнение (поиск) каталога, о котором идет речь. Новая среда реально не вступит в силу, пока вы не выполните вызов `chdir()` в вашей новой среде. Следует также отметить, что процесс может с легкостью выйти из chroot-среды, если он имеет привилегии администратора. Это может быть достигнуто созданием файлов устройств для чтения памяти ядра, подключением отладчика к процессу вне узницы и многими другими способами.

Поведение системного вызова `chroot()` можно некоторым образом контролировать `sysctl` -переменной kern.chroot_allow_open_directories. Когда эта переменная установлена в 0, `chroot()` не сработает с ошибкой EPERM, если есть какие-либо открытые каталоги. Если она установлена в значение по умолчанию, равное 1, то `chroot()` не сработает с ошибкой EPERM, если есть какие-либо открытые каталоги и процесс уже подвергнут вызову `chroot()`. Для всех других значений проверка открытости каталогов будет полностью опущена.

5.6.1. Функциональность джейлов (jail) во FreeBSD

Концепция джейлов (Jail) расширяет возможности `chroot()`, ограничивая власть

администратора созданием настоящих `виртуальных серверов'. Как только тюремная камера создана, все сетевые коммуникации должны осуществляться через выделенный адрес IP, а сила "привилегий пользователя root" в этой тюрьме довольно ограничена.

При работе внутри тюрьмы, любые проверки силы администратора в ядре при помощи вызова `suser()` будут оканчиваться неудачно. Однако некоторые вызовы к `suser()` были изменены на новый интерфейс `suser_xxx()`. Эта функция отвечает за распознание и разрешение доступа к власти администратора для процессов, не находящихся в неволе.

Процесс администратора внутри среды джейла имеет право:

- Манипулировать привилегиями с помощью `setuid`, `seteuid`, `setgid`, `setegid`, `setgroups`, `setreuid`, `setregid` и `setlogin`
- Устанавливать ограничения на использование ресурсов при помощи `setrlimit`
- Модифицировать некоторые sysctl-переменные (kern.hostname)
- `chroot()`
- Устанавливать следующие флаги на vnode: `chflags`, `fchflags`
- Устанавливать такие атрибуты vnode, как права доступа к файлу, изменять его владельца, группу, размер, время доступа и модификации.
- Осуществлять привязку к привилегированному порту в области портов Интернет (порты с номерами 1024)

`Jail` является очень полезным инструментом для запуска приложений в защищенном окружении, но есть и некоторые недостатки. На текущий момент к формату `suser_xxx` не преобразованы механизмы IPC, так что такие приложения, как MySQL, не могут работать в джейле. Права администратора могут имеет малую силу внутри джейла, но нет способа определить, что значит "малую".

5.6.2. POSIX®.1e возможности процессов

POSIX® выпустила рабочий документ, который добавляет аудит событий, списки управления доступом, тонко настраиваемые привилегии, метки информации и жесткое управление доступом.

Этот документ находится в работе и находится в центре внимания проекта TrustedBSD. Некоторая начальная функциональность уже была добавлена во FreeBSD-CURRENT (cap_set_proc(3)).

5.7. Доверие

Приложение никогда не должно полагать, что среда пользователя безопасна. Сюда включается (но этим не ограничено): ввод пользователя, сигналы, переменные среды, ресурсы, IPC, отображаемая в файл память (mmap), рабочий каталог файловой системы, дескрипторы файлов, число открытых файлов и прочее.

Никогда не думайте, что сможете предусмотреть все формы неправильного ввода, который может дать пользователь. Вместо этого ваше приложение должно осуществлять

позитивную фильтрацию, пропуская только конечное множество возможных вариантов ввода, которые вы считаете безопасными. Неполная проверка данных была причиной многих нарушений защиты, особенно CGI-скриптов на веб-сайтах. Для имен файлов вам нужно уделять особое внимание путям ("../", "/"), символическим ссылкам и экранирующим символам оболочки.

В Perl имеется такая очень полезная вещь, как "безупречный" (taint) режим, который можно использовать для запрещения скриптам использовать данные, порожденные вне программы, не безопасным способом. Этот режим проверяет аргументы командной строки, переменные окружения, информацию локализации, результаты некоторых системных вызовов (`readdir()`, `readlink()`, `getpwxxx()` и весь файловый ввод.

5.8. Неожиданное поведение

Неожиданное поведение - это аномальное поведение, вызванное непредусмотренной зависимостью от относительной последовательности событий. Другими словами, программист неправильно предположил, что некоторое событие всегда случается перед другим.

Некоторые из широко распространенных причин возникновения таких проблем являются сигналы, проверки доступа и открытия файлов. Сигналы по своей природе являются асинхронными событиями, так что по отношению к ним нужно проявлять особое внимание. Проверка доступа функцией `access(2)` с последующим вызовом `open(2)` полностью не атомарно. Пользователи могут переместить файлы в промежутке между двумя вызовами. Вместо этого привилегированное приложение должно выполнить `seteuid()`, а затем сразу вызвать `open()`. В тех же строках приложение должно всегда устанавливать явно маску прав доступа (umask) перед вызовом функции `open()` во избежание беспорядочных вызовов `chmod()`.

Часть III: Ядро

Глава 6. История ядра Unix

Немного истории о ядре Unix/BSD, системных вызовах, как работают процессы, блокировке, планировке задач, нити (ядра), переключение контекста, сигналы, прерывания, модули и так далее.

Часть IV: Память и виртуальная память

Глава 7. Виртуальная память

VM, постраничная подкачка и свопирование, выделение памяти, тестирование ошибок утечки памяти, mmap, vnode и так далее.

Часть V: Система ввода/вывода

Глава 8. UFS

UFS, FFS, Ext2FS, JFS, inodes, buffer cache, labeling, locking, metadata, soft-updates, LFS, portalfs, procfs, vnodes, memory sharing, memory objects, TLBs, caching

Часть VI: Межпроцессное взаимодействие

Глава 9. Сигналы

Сигналы, конвейеры, семафоры, очереди сообщений, совместно используемая память, сокеты, двери

Часть VII: Работа в сети

Глава 10. Сокеты

Сокеты, bpf, IP, TCP, UDP, ICMP, OSI, bridging, firewalling, NAT, коммутация и так далее

Часть VIII: Сетевые файловые системы

Глава 11. AFS

AFS, NFS, SANs etc]

Часть IX: Работа с терминалами

Глава 12. Системные консоли

Syscons, tty, PCVT, последовательная консоль, хранители экрана и так далее

Часть X: Звук

Глава 13. OSS

OSS, waveforms, etc

Часть XI: Драйверы устройств

Глава 14. Устройства USB

Эта глава расскажет о механизмах, используемых во FreeBSD для написания драйверов для устройств на шине USB.

Глава 15. NewBus

Эта глава расскажет об архитектуре NewBus во FreeBSD.

Часть XII: Аппаратные платформы

Глава 16. IA-32

Рассказ об архитектурных особенностях FreeBSD/x86.

Глава 17. Alpha

Рассказ об архитектурных особенностях FreeBSD/alpha.

Описание ошибок выравнивания, как их исправлять и как игнорировать.

Пример ассемблерного кода для FreeBSD/alpha.

Глава 18. IA-64

Рассказ об архитектурных особенностях FreeBSD/ia64.

Часть XIII: Отладка

Глава 19. Truss

Различные описания того, как отлаживать отдельные компоненты системы при помощи утилит truss, ktrace, gdb, kgdb, etc

Часть XIV: Обеспечение совместимости

Глава 20. Linux

Linux, SVR4 и так далее

Приложения

[1] Dave A Patterson and John L Hennessy. Copyright® 1998 Morgan Kaufmann Publishers, Inc. 1-55860-428-6. Morgan Kaufmann Publishers, Inc. Computer Organization and Design. The Hardware / Software Interface. 1-2.

[2] W. Richard Stevens. Copyright® 1993 Addison Wesley Longman, Inc. 0-201-56317-7. Addison Wesley Longman, Inc. Advanced Programming in the Unix Environment. 1-2.

[3] Marshall Kirk McKusick and George Neville-Neil. Copyright® 2004 Addison-Wesley. 0-201-70245-2. Addison-Wesley. The Design and Implementation of the FreeBSD Operating System. 1-2.

[4] Aleph One. Phrack 49; "Smashing the Stack for Fun and Profit".

[5] Chrispin Cowan, Calton Pu, and Dave Maier. StackGuard; Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks.

[6] Todd Miller and Theo de Raadt. strlcpy and strlcat — consistent, safe string copy and concatenation.