«Технологии QNX и КПДА в России»

20 ноября 2019, Санкт-Петербург

Приёмы безопасного программирования при разработке ПО для ЗОСРВ «Нейтрино»

Олег Большаков, ООО «СВД Встраиваемые Системы»

СВД Встраиваемые Системы

Нормативные документы

- **Приказ ФСТЭК России от 14 марта 2014 г. № 31** Об утверждении требований к обеспечению защиты информации в автоматизированных системах управления производственными и технологическими процессами на критически важных объектах, потенциально опасных объектах, а также объектах, представляющих повышенную опасность для жизни и здоровья людей и для окружающей природной среды

- **Положение о сертификации средств защиты информации** (утв. приказом ФСТЭК России от 3 апреля 2018 г. № 55)

- **ГОСТ Р 56939-2016** Защита информации. Разработка безопасного программного обеспечения. Общие требования

- **ГОСТ Р 58412-2019** Защита информации. Разработка безопасного программного обеспечения. Угрозы безопасности информации при разработке программного обеспечения

СВД Встраиваемые Системы

Философия языка C

(a) Доверять программисту.
(b) Не мешать программисту делать то, что должно.
(c) Поддерживать язык миниатюрным и простым.
(d) Предоставлять единственный способ выполнения операции.
(e) Скорость превыше портируемости.

Аспекты (a) и (b) напрямую противоречат безопасности и защищённости.

❑ Программист должен сам заботиться о безопасности!

Надёжный, уязвимый, безопасный

Надёжный код — устойчивый к сбоям код:

- ❑ Предотвращение аварийного завершения и неожиданного поведения
- ❑ Корректная обработка некорректного ввода
- ❑ Выявление и корректная обработка внутренних ошибок
  - Предоставление полезной информации при сбоях для анализа и восстановления
- ❑ Исключение аварийных ситуаций (корректное завершение)

Уязвимый код — любой ***не*** надёжный код

Безопасный код — отвечает требованиям политики безопасности

- Неявно накладывает требования надёжности

СВД Встраиваемые Системы

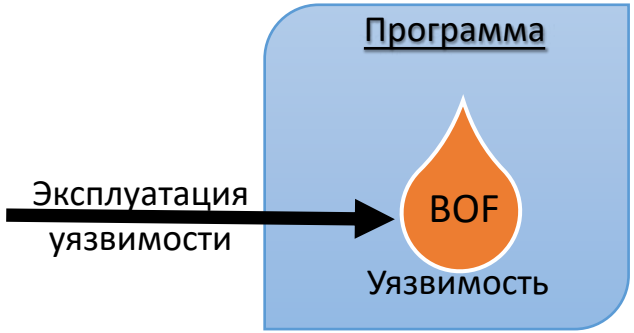
Надёжный или безопасный

Безопасная программа — отвечает требованиям политики безопасности

- Неявно накладываются требования надёжности

Надёжная программа делает то, что требуется, рационально обрабатывает непредвиденные ситуации и не завершается «падением»

- Может завершаться корректно



Получены дополнительные привилегии:
Не безопасная программа

Не получены дополнительные привилегии:
Безопасная программа,
но не надёжная программа

Где искать уязвимости?

- Программа изменяет уровень привилегий
  - Атрибуты setuid, setgid
- Предположение об атомарности операций
  - Проверка атрибутов доступа, а затем открытие файла
- Доверие окружению
  - Предположение, что программа запущена в том окружении, для которого собрана
  - Предположение, что ввод корректен и верно отформатирован

- Слышь, Толян, а здесь медведи есть?
- Какие ещё, нафиг, медведи!..

Что следует проверять?

При чтении данных, которые не контролируются, следует **всегда** выполнять проверку достоверности

- Работа с буферами — проверка размера данных
- Работа с числами — проверка знака и величины
- Для сетевых данных — проверка соответствия требованиям RFC
  - Например, для DNS недопустимы символы — ; * *<пробел>*
- Для чтения строк — не использовать `*scanf()` и `gets()`

И не только данных...

- Проверять соответствие прав файлов и пользователя

Пример: Heartbleed Bug

Проблема: библиотека OpenSSL некорректно обрабатывает пакет Heartbteed (*CVE-2014-0160*).

- Пакет Heartbleed используется для проверки доступности сервиса
- Пакет содержит: размер данных, сами данные и данные выравнивания
- Сервер возвращает данные обратно

Атака:

- Сформировать пакет с небольшой нагрузкой с размером данных больше реальной нагрузки (до $2^{16} - 1 = 65535$)
- Отправить пакет «жертве»
- OpenSSL не проверяет длину пакета, доверяя данным из пакета
- OpenSSL возвращает нагрузку и другие данные, которые попали в буфер
  - Данные могут содержать пароли или «cookie»

Переменные окружения

Переменные окружения контролирует пользователь.

- Явное использование:

```
p = getenv("HOME");
```

- Неявное использование:

```
system("echo -n 'Today is '; date");
```

- Переменные окружения контролирует пользователь
- Следует проверять значение переменных окружения
- **Лучше** установить переменные в безопасные значения

Пример: Излишняя отладочная информация

Проблема: `sendmail` запускается с битом setuid от root (*CVE-2001-0713*)

- -C используется для проверки и отладки конфигурационного файла
  - Отличная диагностика, включая вывод некорректной строки

Атака:

`sendmail -C <секретный_файл>`

Получаем доступ к любому файлу в системе.

Очевидное, но **неправильное** решение:

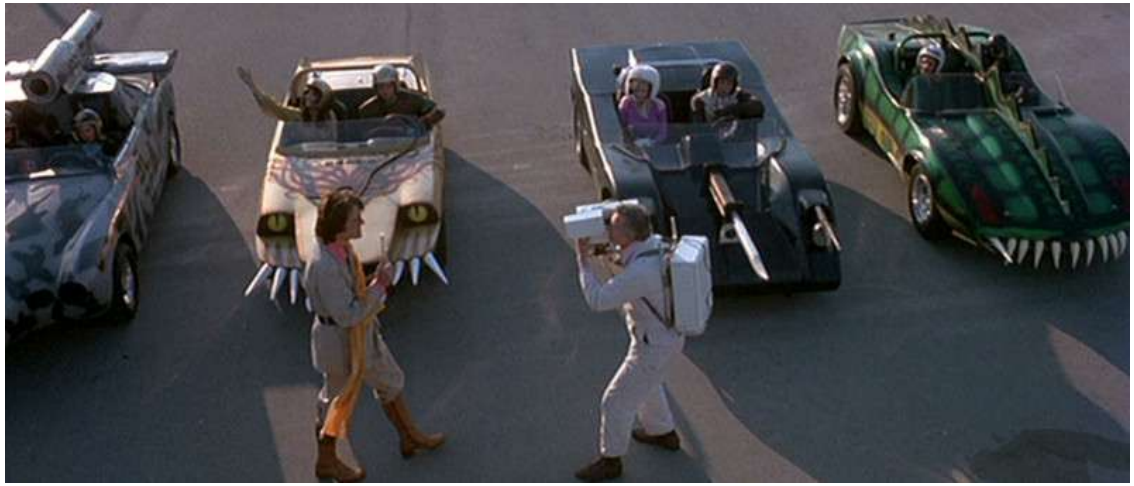
```
if (access(configfile, R_OK) == 0) {
    fp = fopen(configfile, "r");
    /* использование fp */
```

Такое решение приводит к гонкам…

Гонки

- Состояние гонки возникает, если есть два события A и B
  - Порядок AB ожидаемый
  - Порядок BA приводит к проблеме
  - A и B участвуют в гонке, чтобы финишировать (произойти) раньше

- Состояние гонки возникает в многопоточной или многозадачной среде
  - Два процесса или потока взаимодействуют
  - Один из них «жертва», другой — нарушитель



Пример: Гонки

Проблема:

```
if (access("file", R_OK) == 0) {
    fp = fopen("file", "r");
    /* использование fp */
```

Неправильно:

```
fp = fopen("file", "r");
if (access("file", R_OK) == 0) {
    /* использование fp */
```

Правильно:

```
fp = fopen("file", "r");
if (fstat(fileno(fp), &stbuf) < 0)
    close(fp);
/* проверка владельца, группы и т.д. */
```

Пример: Кто проверяет?

Классический пример `rexd` (*CVE-1999-0627*)

- Предположение — клиент всё проверяет:
  - Аутентификацию пользователя
  - Авторизацию команд

- Допущение — `rexd` может не выполнять проверок!

Ошибка: кто угодно может написать клиент для вашего сервера

Как надо:

- **Обязательно** выполнять проверки на стороне сервера
- Можно выполнять проверки на стороне клиента

Внедрения

- Внедрения происходят в случае встраивания данных, команд или инструкций в ожидаемый ввод
  - Это приводит к непредусмотренной интерпретации введённых данных

Примеры типов внедрений:

- Веб (например, межсайтовые сценарии)
- Внедрение SQL-кода
- Внедрение команд

Внедрение команд — подача на вход данных, которые преобразуют команду в другую команду с целью выполнения вредоносных действий

Injection (англ.) — инъекция, внедрение

Пример: Внедрение команд

Код привилегированной программы:

```
if (filename = getuserfile(stdin)) == NULL)
    return -1;
(void)sprintf(cmd, 1024, "cat data > %s", filename);
system(cmd);
```

Пользователь вводит:

```
/tmp/file1; cat secret > /tmp/file2
```

Защищаемый файл `secret` будет скопирован в `/tmp/file2`

Вариант решения:

- «Чёрный список» команд/символов — **ненадёжно**

Как надо:

- «Белый список» допустимых команд/символов

Дочерние процессы

- Создание — `fork()` — копия основного процесса со своим PID
- Наследуют окружение включая файловые дескрипторы

Угроза:

- Привилегированный процесс открыл защищаемый файл
- Привилегированный процесс создал дочерний процесс
  - При этом родительский процесс сбросил привилегии

- Непривилегированный процесс может читать данные из дескриптора:

```
fd = open("secret", O_RDONLY);
dup2(fd, 9);
system("/bin/sh");
```

Получить доступ к содержимому файла:

```
$ cat <&9
```

Особенности открытия файлов

- Уровень доступа проверятся только при открытии
  - Не проверяется при чтении, записи и т.д.

- Удобно использовать для каналов и журналов
  - Защищаемый файл открывается с правами суперпользователя
  - Сбрасываются привилегии
  - Файл остаётся доступным

Если файл не должен быть доступен, то **как надо:**

- Установить флаг «закрыть при исполнении»

```
fcntl(9, F_SETFD, FD_CLOEXEC);
ioctl(9, FIOCLEX, NULL);
open("sectret", O_RDONLY | O_CLOEXEC);
```

Строки: Виды ошибок

- Некорректное копирование статических строк — `gets()`, `strcpy()/strcat()`, `sprintf()`

```
char buf[128];
gets(buf);
```

- Ошибка завышения на единицу

```
buf = malloc(strlen(src));
for (i = 0; i < strlen(src); i++)
    buf[i] = src[i];
buf[i] = '\0';
```

- Ошибка завершения нулём

```
char buf[16];
strncpy(buf, "0123456789abcdef", 16);
```

- Ошибка усечения

Безопасный ввод

```
char buf[LINE_MAX];
int ch;
char *p;

if (fgets(buf, sizeof(buf), stdin)) {
    p = strchr(buf, '\n'); /* Ищем и удаляем перевод строки */
    if (p)
        *p = '\0';
    else {
        /* Перевод строки не найден, дочитываем строку */
        while (((ch = getchar()) != '\n') && !feof(stdin) &&
                !ferror(stdin)) ;
    }
} else {
   /* Обработка ошибки fgets() */
}
```

Безопасное копирование/объединение

Распространённая причина переполнения буфера (BOF):

```
char * strcpy (char *dst, const char *src);
char * strcat (char *dst, const char *src);
```

Рекомендация C99:

```
char * strncpy(char *dst, const char *src, size_t n);
char * strncat(char *dst, const char *src, size_t n);
```

Безопасная реализация OpenBSD:

```
size_t strlcpy(char *dst, const char *src, size_t size);
size_t strlcat(char *dst, const char *src, size_t size);
```

Сравнение функций копирования/объединения

	Стандарт	Защита от переполнения буфера	Гарантированное завершение нулём	Усечение строки	Динамическое выделение памяти
`strcpy()`	C99	Нет	Нет	Нет	Нет
`strncpy()`	C99	Да	Нет	Да	Нет
`strlcpy()`	OpenBSD	Да	Да	Да	Нет
`strdup()`	TR 24731-2	Да	Да	Нет	Да

	Стандарт	Защита от переполнения буфера	Гарантированное завершение нулём	Усечение строки	Динамическое выделение памяти
`strcat()`	C99	Нет	Нет	Нет	Нет
`strncat()`	C99	Да	Нет	Да	Нет
`strlcat()`	OpenBSD	Да	Да	Да	Нет

Меры повышения качества

Для выявления потенциальных уязвимостей необходимо:

- ❑ Регулярно проводить статический анализ исходного кода
`cppcheck`

- ❑ Отслеживать и устранять предупреждения компилятора
`gcc -Wall -Wextra -pedantic …`

- ❑ Регулярно выполнять модульное тестирование
`cunit`
`cppunit`
`GoogleTest`

- ❑ Применять средства динамического инструментирования и отладки
`gcc -fsanitize=address`
`Valgrind`

Последняя линия обороны

- Защита от разрушения стека (технология SSP)

```
gcc -fstack-protector*
```

- Защита структуры исполняемого файла

```
gcc -Wl,"-z relro" -Wl,"-z now"
```

- Защита стека от исполнения

```
startup-bios -x
```

- Рандомизация адресного пространства (технология ASLR)

```
procnto* -mr
```

Спасибо за внимание

Олег Большаков
Руководитель группы разработки
ООО «СВД Встраиваемые Системы»

(812) 346-89-56 доб. 108
o.bolshakov@kpda.ru
www.kpda.ru
forum.kpda.ru
vk.com/cbd_bc
fb.com/cbdbc
kpda_info

СВД Встраиваемые Системы