О методах динамического встраивания в ядро операционной системы

(на примере ядра Linux)

Матвейчиков Илья

21 августа 2014 г.

О докладчике

Матвейчиков Илья Владимирович

- МИФИ, 2005
- ООО «Код Безопасности»
- Разработка средств защиты информации
  - в том числе, криптографической (СКЗИ)

Области интересов

- Системное программирование
- Компьютерная безопасность
- Reverse Code Engineering
- Linux Kernel

Код безопасности

О докладе

Целевая аудитория

- Системные разработчики
- Специалисты по безопасности
- Все, кому это может быть интересно :)

Предметная область

- Ядро Linux
- Архитектура x86 (x86_64)
- Программирование на C/ASM

Код безопасности

О встраивании

Встраивание – процесс внедрения дополнительных (сторонних) элементов, осуществляемый при сохранении работоспособности целевой системы.

Цели и задачи встраивания

- контроль, модификация и расширения функций компонентов программной системы
- обеспечение внедрения в существующую программную систему

Как правило, при встраивании должно обеспечиваться сохранение работоспособности целевой системы.

Код безопасности

Примеры встраивания

Динамическое встраивание – процесс встраивания, без необходимости перезагрузки целевой системы.

Встраиванием является

- установка обновления (ksplice, kpatch, kGraft)
- подгрузка модуля ядра (при подключении нового оборудования)
- внедрение зловреда при эусплуатации уязвимости

Наложение патча на ядро с последующей сборкой не является динамическим, т.к. требует перезагрузки.

Код безопасности

Традиционные методы встраивания

Направления, являющиеся наиболее распространённым:

- перехват функций ядра
- перехват системных вызовов
- перехват обработчиков исключений

Основой для встраивания является патчинг - техника внесения изменений в код или данные, позволяющая модифицировать поведение целевого алгоритма требуемым образом.

Первое упоминание патчинга ядра Linux - Silvio Cesare, 1999 год.

Код безопасности

Традиционные методы встраивания
Перехват функций ядра

Преимущество

Возможность контролировать поведение ядра ОС с точностью до отдельных функций.

Функция inode_permission (fs/namei.c)

- контроль доступа субъекта (пользователя) к объекту (файлу) файловой системы
- ключевая функция проверки полномочий
- пошалим?..

Код безопасности

Традиционные методы встраивания
Перехват функций ядра (подробности)

Создание файла

$vfs\_create \rightarrow may\_create \rightarrow inode\_permission$

Техника осуществления перехвата

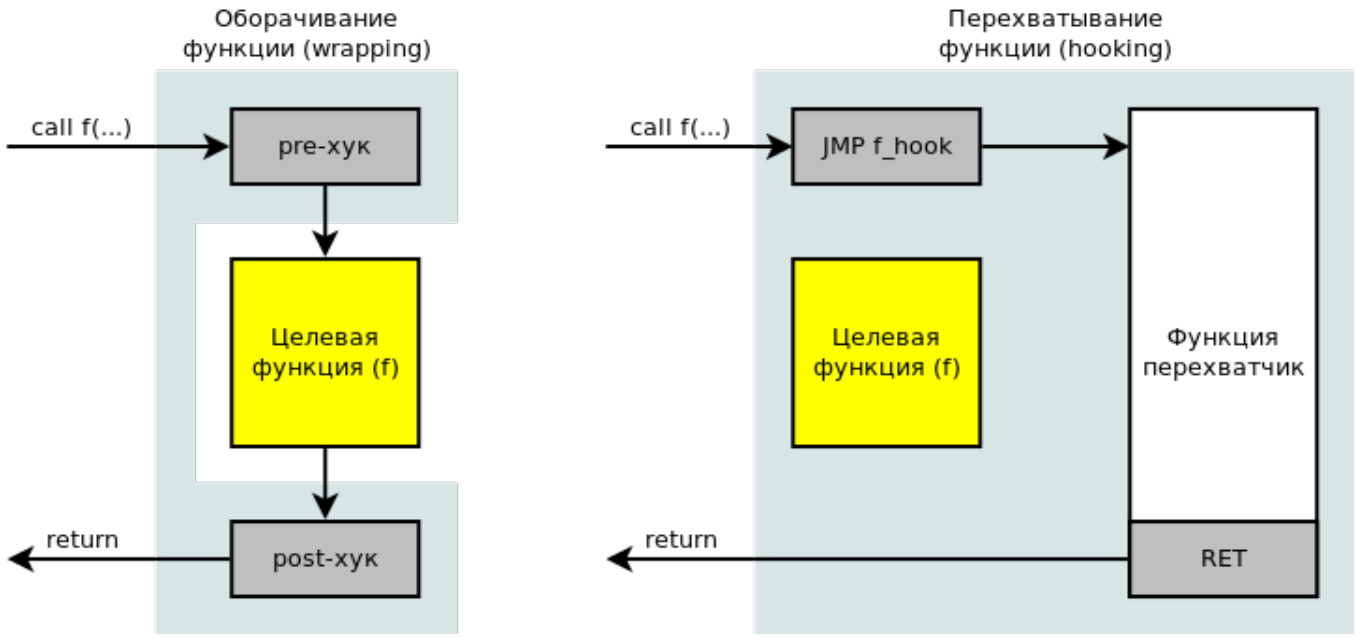
- целевая функция – *inode_permission*
- встраивание с использованием патчинга пролога
- использование обёрток (wrapper)

Код безопасности

Традиционные методы встраивания

Перехват функций ядра (подробности)

Перехват функций и их оборачивание

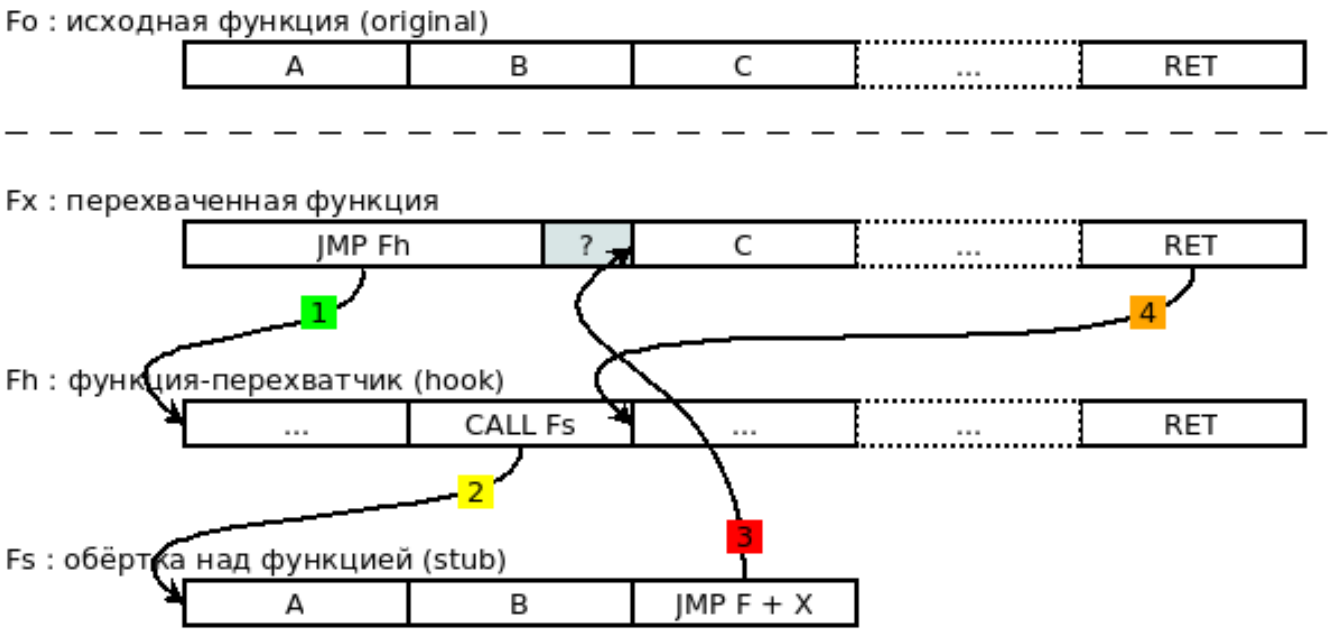


Код безопасности

Традиционные методы встраивания

Перехват функций ядра (подробности)

Модификация пролога и создание функции-обёртки



Код безопасности

Традиционные методы встраивания
Перехват функций ядра (подробности)

Базовый метод перехвата - патчинг пролога функции.

Особенности реализации

- Необходимость сохранения пролога
  - Использовать дизассемблер (как минимум, LDE)
- Необходимость синхронизации на SMP
  - На время модификации «парковать» все потоки, кроме одного, осуществляющего модификацию (*stop_machine*)
  - Обращать внимание на содержимое кеша процессора, после модификации
- Необходимость обхода страничной защиты
  - отключать или обходить страничную защиту

Код безопасности

Традиционные методы встраивания

Перехват диспетчера системных вызовов

Преимущество

Возможность контролировать запросы прикладного ПО к сервисам ядра ОС (инспекция запросов, аудит).

Создание файла

- man 2 open
- $USER \Rightarrow KERNEL(system\_call)$

Код безопасности

Традиционные методы встраивания
Перехват диспетчера системных вызовов (подробности)

Создание файла

$system\_call \rightarrow sys\_call\_table[\_\_NR\_open](args..) \rightarrow sys\_open$

Диспетчеризация

- диспетчер системных вызовов (system_call)
- таблица системных вызовов (sys_call_table)
- функция системного вызова (sys_open)

Код безопасности

Традиционные методы встраивания
Перехват диспетчера системных вызовов (подробности)

Способы осуществления перехвата

- модификация диспетчера
  - единая точка встраивания
  - позволяет перехватить все системные вызовы
  - полный контроль над входными и выходными параметрами
- модификация или подмена таблицы
  - множественное встраивание, т.к. одному элементу таблицы соответствует один системный вызов
  - позволяет перехватить все системные вызовы
- перехват конкретной функции
  - множественное встраивание, т.к. одной функции соответствует один системный вызов
  - позволяет перехватить многие системные вызовы (есть особенности)

Код безопасности

Традиционные методы встраивания
Перехват диспетчера системных вызовов (подробности)

Перехват диспетчера исключений - универсальный и наиболее эффективный метод встраивания.

Техника перехвата диспетчера

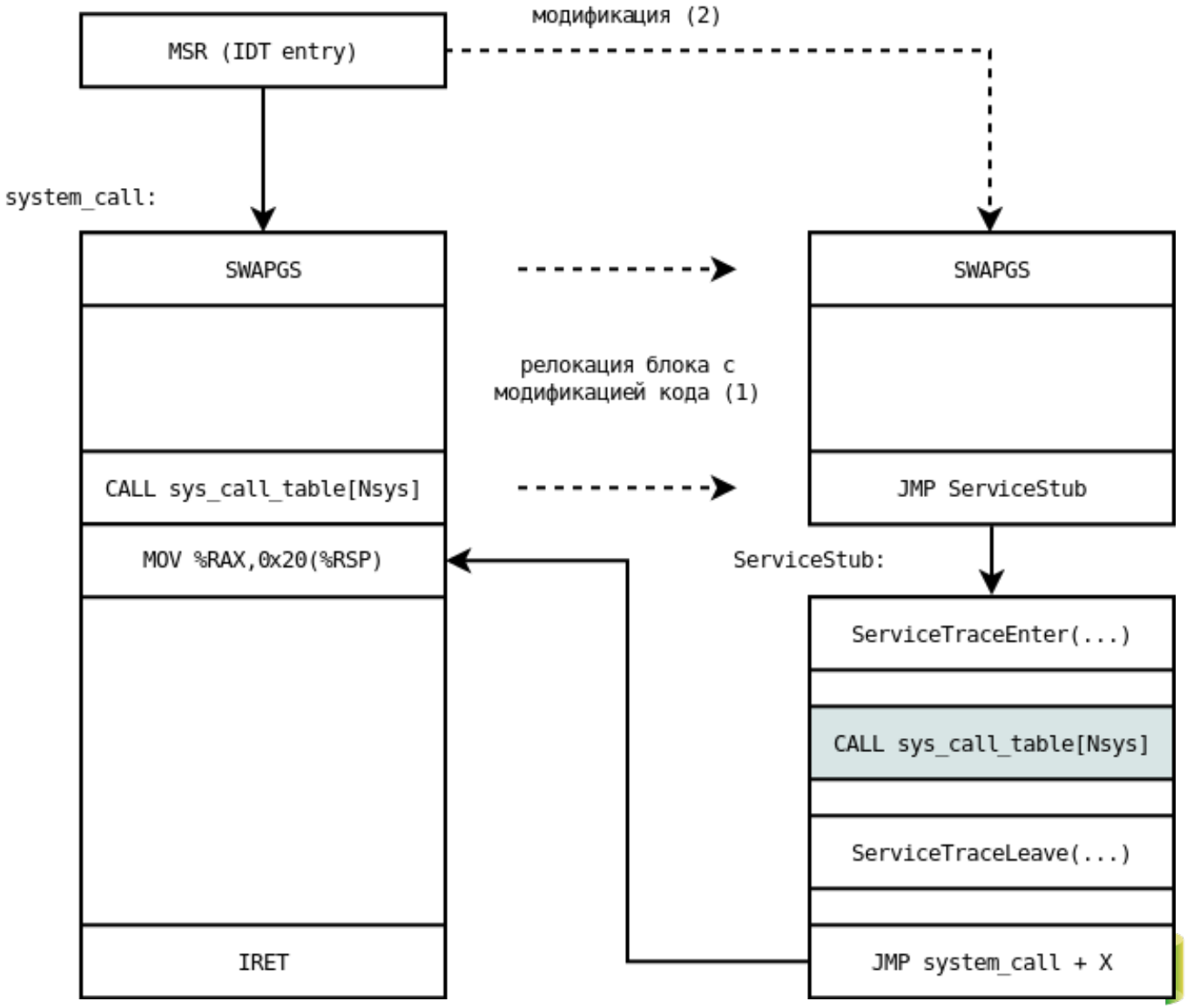
- возможны варианты
  - с модификацией кода (патчинг и/или сплайсинг)
  - без модификации кода (замещение вектора в IDT, перенастройка регистров MSR и т.д.)

Сплайсинг - разновидность патчинга, не нарушающая логическую целостность кода. Как правило, целью сплайсинга бывает осуществление модификации адреса перехода команды.

Код безопасности

Традиционные методы встраивания

Перехват диспетчера системных вызовов (подробности)



Традиционные методы встраивания
Перехват диспетчера системных вызовов (подробности)

Базовый метод перехвата - патчинг или подмена адресов обработчиков в IDT (MSR).

Особенности реализации

- Необходимость разбора структуры диспетчера
- Необходимость учёта разновидностей диспетчеров от ядра к ядру и от вендора к вендору :)

Метод релокации кода - хорошая идея создать свою копию части диспетчера для дальнейшей модификации. С пре- и пост- хуками, конечно :)

Код безопасности

Традиционные методы встраивания

Перехват обработчиков исключений

Возможность контролировать ключевые механизмы ядра ОС (например, обработчик PageFault (#PF) или INT3 (#BP).

Обработка INT3

$INT3 \rightarrow Exception(\#BP) \rightarrow do\_trap \rightarrow do\_int3$

INT3 - хороший способ перехвата функций, т.к. размер команды – один байт.

Код безопасности

Традиционные методы встраивания
Перехват обработчиков исключений (подробности)

Перехват обработчиков исключений - хороший способ встроиться в ключевые системные механизмы.

Техника перехвата обработчиков

- возможны варианты
  - с модификацией кода (патчинг и/или сплайсинг)
  - без модификации кода (замещение вектора в IDT, перенастройка регистров MSR и т.д.)

Код безопасности

Традиционные методы встраивания

Перехват обработчиков исключений (подробности)

Типичная структура обработчика исключения (на примере обработчика INT3):

```
int3:
..d070: ff 15 ea 1b 19 00          callq *0x191bea(%rip)
..d076: ...
// паразитный CALL (нужно пропускать)
..d07c: e8 3f ff ff ff             callq ffffffff812fcfc0
..d081: ...
// вызов подпрограммы обработки (do_int3)
..d09e: e8 8d 06 00 00             callq 0xffffffff812fd730
..d0a3: ...
```

Целью модификации является команда `callq 0xffffffff812fd730`. К ней применяется сплайсинг и в результате управление получает любой перехватчик.

Код безопасности

Традиционные методы встраивания
Перехват обработчиков исключений (подробности)

Базовый метод перехвата - патчинг или подмена адресов обработчиков в IDT (MSR).

Особенности реализации

- Необходимость разбора структуры диспетчера

Метод релокации кода - создание своей копии части обработчика для дальнейшей модификации.

Код безопасности

Традиционные методы встраивания
Проблемы и способы их решения (1)

Модификация кода на SMP

- использовать stop_machine

Обход страничной защиты

- отключать страничную защиту в CR2 (плохо)
- менять атрибуты страниц в PTE с RX на RWX (лучше)
- создавать отображение страницы с требуемыми атрибутами (хорошо)

Код безопасности

Традиционные методы встраивания
Проблемы и способы их решения (2)

Анализ структуры кода

- Использовать сигнатуры (плохо)
- Использовать дизассемблер длин (лучше)
- Использовать полноценный дизассемблер (хорошо)

Поиск скрытых и неэкспортируемых символов

- Использовать механизм kallsyms
- Использовать /proc/kallsyms (чтение из ядра)
- Использовать сигнатуры (не всегда достаточно)
- Использовать дизассемблер + сигнатуры

Нетрадиционные методы встраивания

Направления, являющиеся наиболее распространённым:

- использование аппаратных точек останова
- использование возможностей системы (встраивание в VFS, LSM, netfilter и т.д.)
- исползование исключений и метод инвалидации данных

Рассматриваемые методы имеют ограниченное применение. Область их применения зависит от исходных ограничений (например, запрет на патчинг кода).

Код безопасности

Нетрадиционные методы встраивания

Аппаратные точки останова

Преимущества

- Без модификации кода
- Аппаратная поддержка перехвата

Ограничения

- Зависимость метода от конкретной платформы
- Ограниченность применимости (на x86 есть только 4 HWBP)
- Необходмость встраивания в обработчик исключения

Аппаратные точки останова - дополнительное средство

Код безопасности

Нетрадиционные методы встраивания
Встраивание с использованием LSM

Linux Security Modules

- расширения к стандартной модели (DAC)
- относительно простая смена моделей (MAC, ролевая и т.д.)
- технически - 100500 хуков в ядре

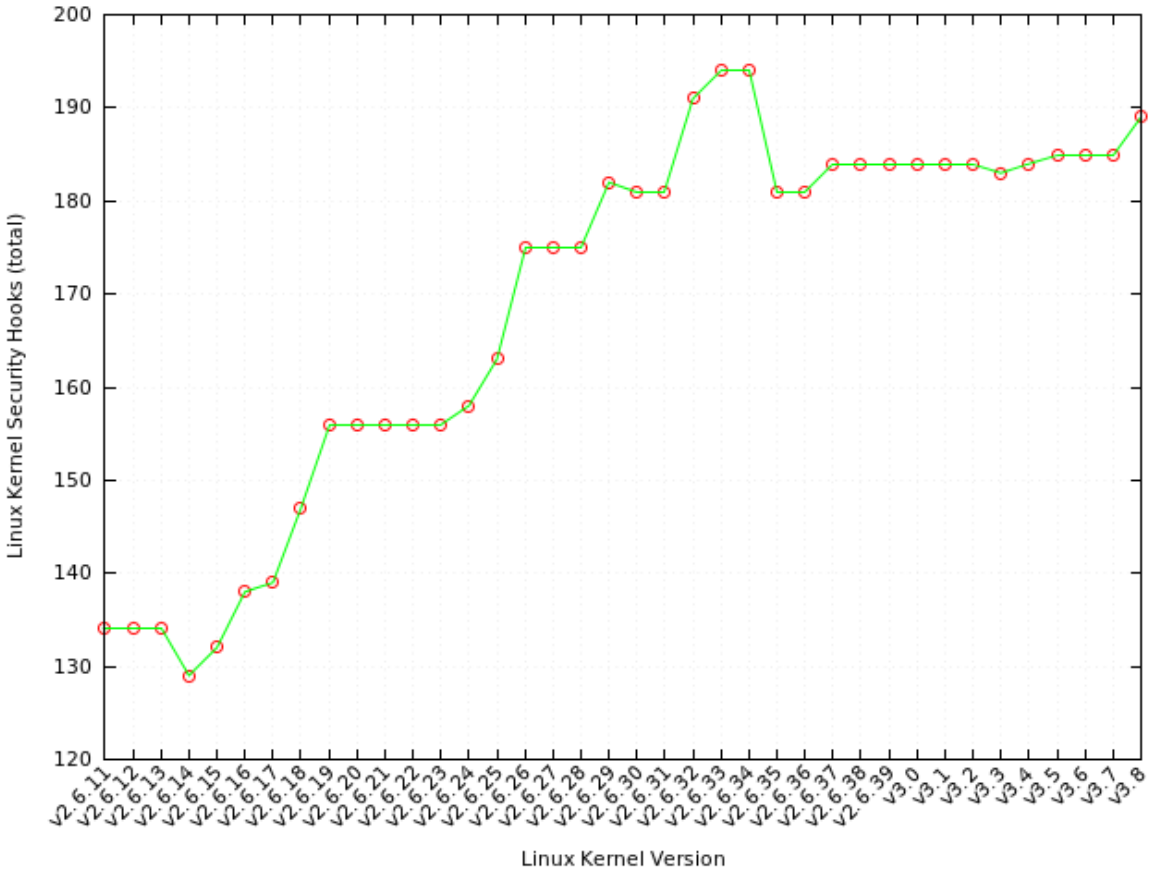
Особенности

- CONFIG_SECURITY
- структура-описатель модели (`struct security_operations`)
- указатель текущей активной модели (`security_ops` → `security_operations`)

Код безопасности

Нетрадиционные методы встраивания

Встраивание с использованием LSM (особенности)



Код безопасности

Нетрадиционные методы встраивания

Встраивание с использованием LSM (особенности)

Иллюстрация использования security_ops кодом ядра:

```
568 int security_inode_permission(struct inode *inode, int mask)
569 {
570         if (unlikely(IS_PRIVATE(inode)))
571                 return 0;
572         return security_ops->inode_permission(inode, mask);
573 }
```

Методика встраивания

- поиск символа security_ops
- замена указателя на структуру модели (SELinux → XXX)

Код безопасности

Нетрадиционные методы встраивания

Метод инвалидации данных

Суть инвалидации (на примере LSM)

```
1) NULL pointer dereference (#PF)
   security_ops = NULL
   security_ops->inode_permission(inode, mask)
2) security_ops = 0xffff800000000000
   security_ops->inode_permission(inode, mask)
   General Protection Fault (#GP)
```

Последовательность

- инвалидация переменной
- обработка исключения
- профит :)

Код безопасности

Раз-раз . . . и в production

SecretNet LSP (c) ООО «Код Безопасности»

- наложенное СЗИ от НСД
- сертифицированное ФСТЭК РФ
- встраиваемые в ядро механизмы обеспечения
  - очистки оперативной памяти
  - очистки (затирания) остаточной информации на диске при удалении файлов
  - аудита действий пользователя (процессов)
  - контроля доступа к устройствам

Код Безопасности

Раз-раз . . . и в production

при разработке SecretNet LSP использовались

- методы перехвата функций ядра (механизмы контроля, очистки)
- методы перехвата диспетчера системных вызовов (механизмы аудита)
- активно использовался патчинг кода ядра :)

Код безопасности

Основные выводы

- Встраивание актуально и востребовано
- Рассмотренные методы надёжны и позволяют решать задачи, связанные в том числе с созданием средств защиты
- Наличие методов встраивания без модификации кода позволяет моделировать возможные атаки на системы контроля целостности компонентов

Код безопасности

Вопросы

Вопросы

Код безопасности