ИСП РАН

Мелкогранулярная рандомизация адресного пространства программы при запуске

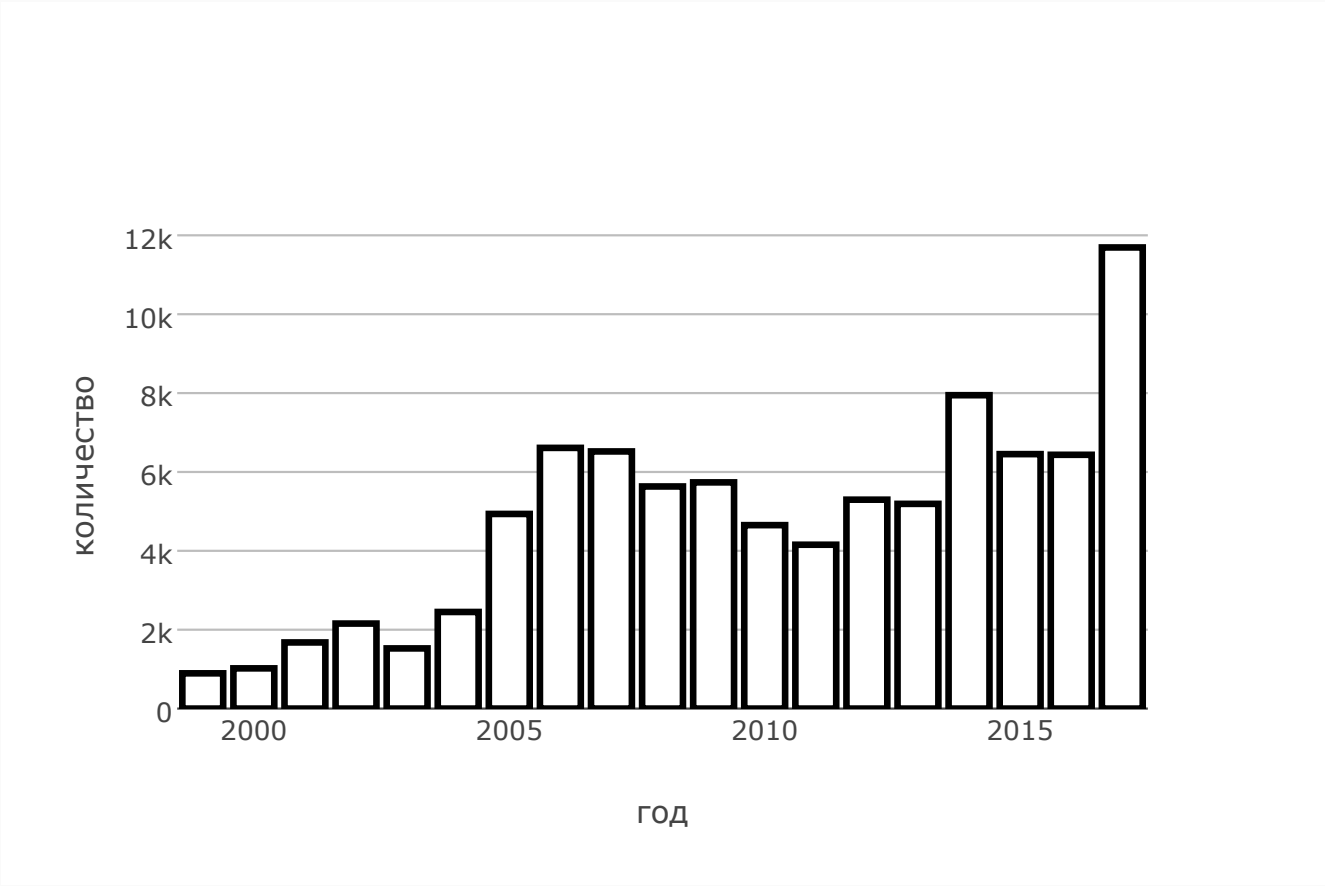
19 ноября 2017 г.

Институт системного программирования РАН

Введение

Актуальность

- Ошибки и уязвимости в ПО неизбежны.



статистика CVE по годам

Актуальность

- Уязвимости всегда будут пытаться эксплуатировать.
- Текущие защиты (DEP, ASLR, PaX) не достаточны.

CVE-2013-1690 использовался ФБР для деанонимизации пользователей Tor.

Постановка задачи

Требуется реализовать мелкогранулярную рандомизацию программ:

- выполняется при запуске,
- производится на уровне функций,
- поддерживает возможность полносистемной развертки
- на Linux x86-64.

Принятые ограничения:

- не требуется перерандомизация во время работы,
- необходим исходный код ПО,
- не поддерживается рандомизация адресного пространства ядра.

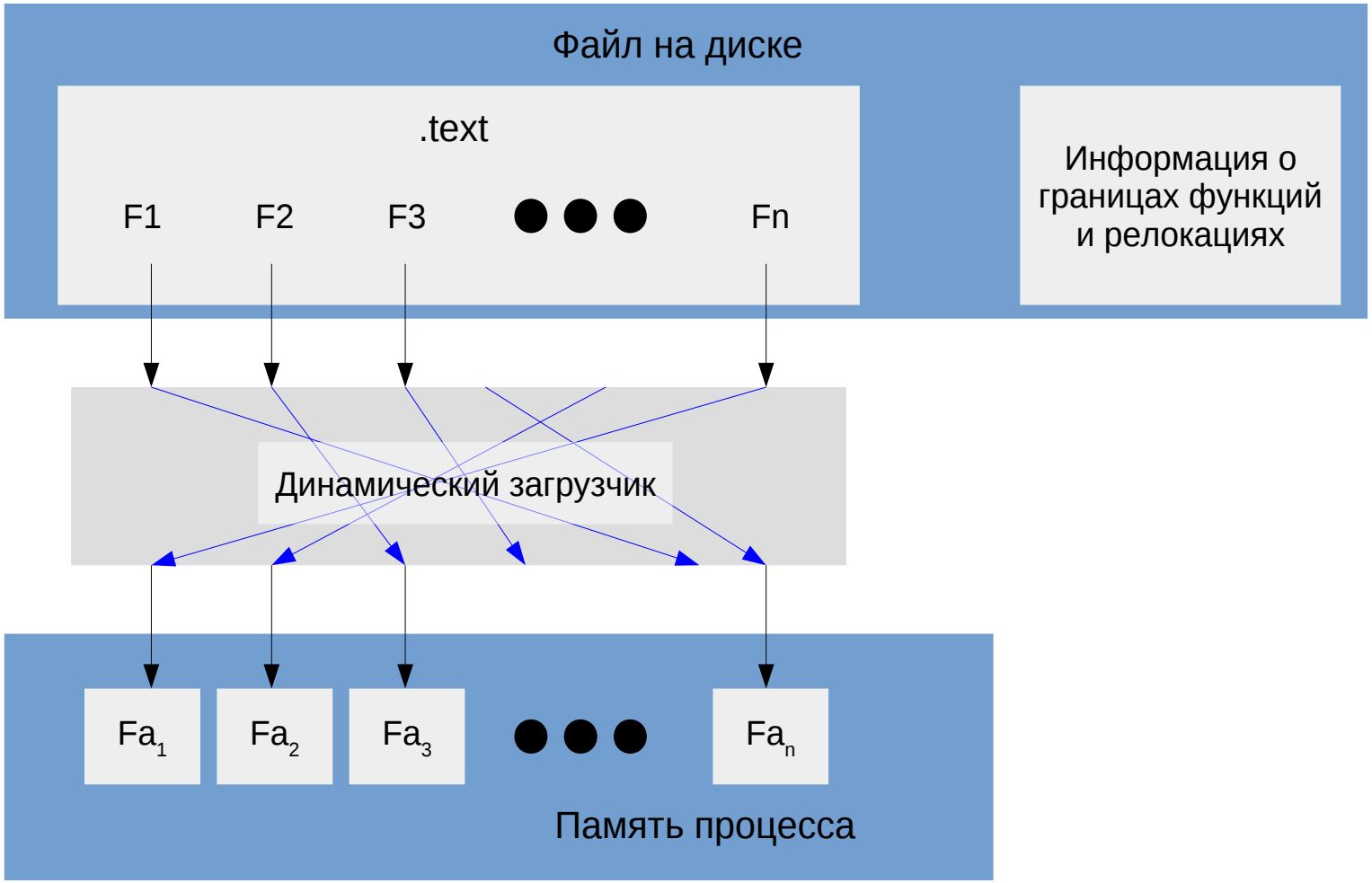
Существующие решения

Существующие решения

- Selfrando
- Oxymoron, Pagerando
- Рандомизация во время работы
- Мелкогранулярная рандомизация во время сборки

Реализация

Схема работы



Компоновка

Создается специальная секция, в которой сохраняется:

- точка входа,
- границы функций (начало, длина и необходимое выравнивание),
- релокации (адрес и тип, а также номера функций в которых содержится целевой адрес и сама релокация).

Загрузка

Поддержка со стороны загрузчика требует:

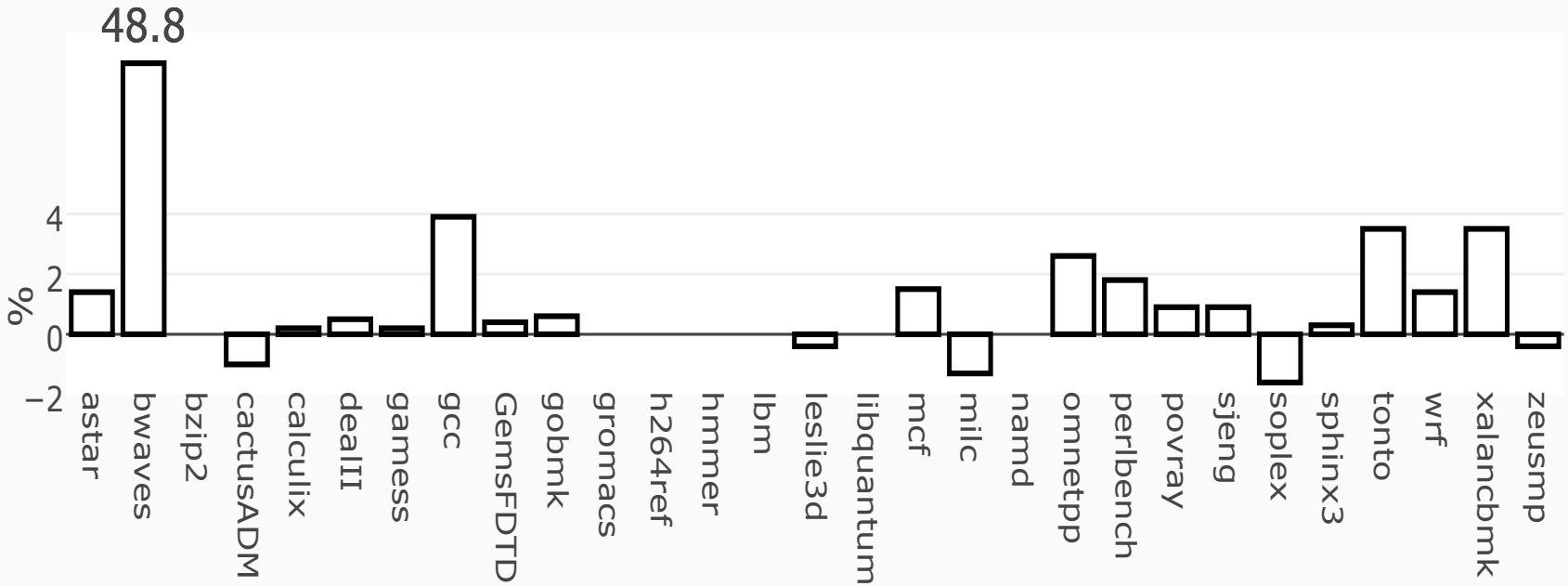
- поиск специальной секции,
- изменение прав доступа RW -> RE,
    - проблема: PaX
- изменение порядка расположения функций.

Тестирование

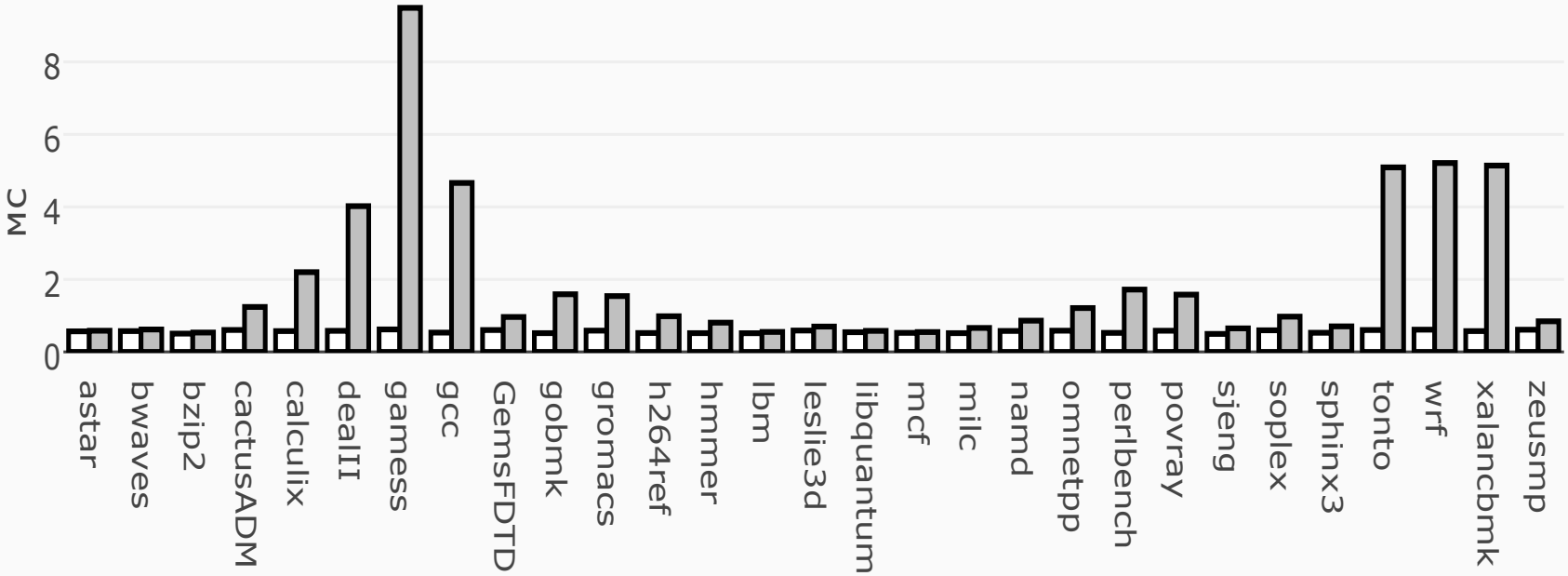
Окружение

- Intel i7-4790
- 16 GB RAM
- CentOS 7
- Linux 3.10 + PaX
- gcc 4.8.5
- binutils 2.23
- glibc 2.17

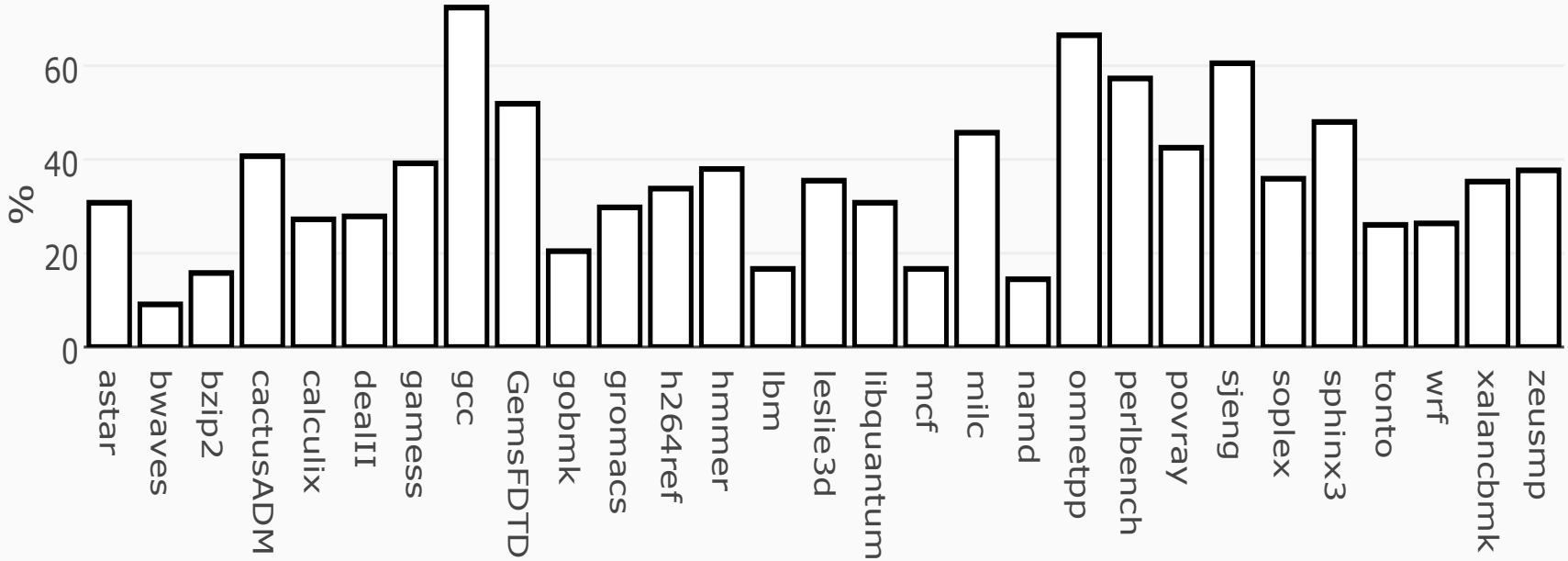
Изменение производительности



Изменение времени старта программы



Изменение размера исполняемого файла



Сравнение с другими инструментами

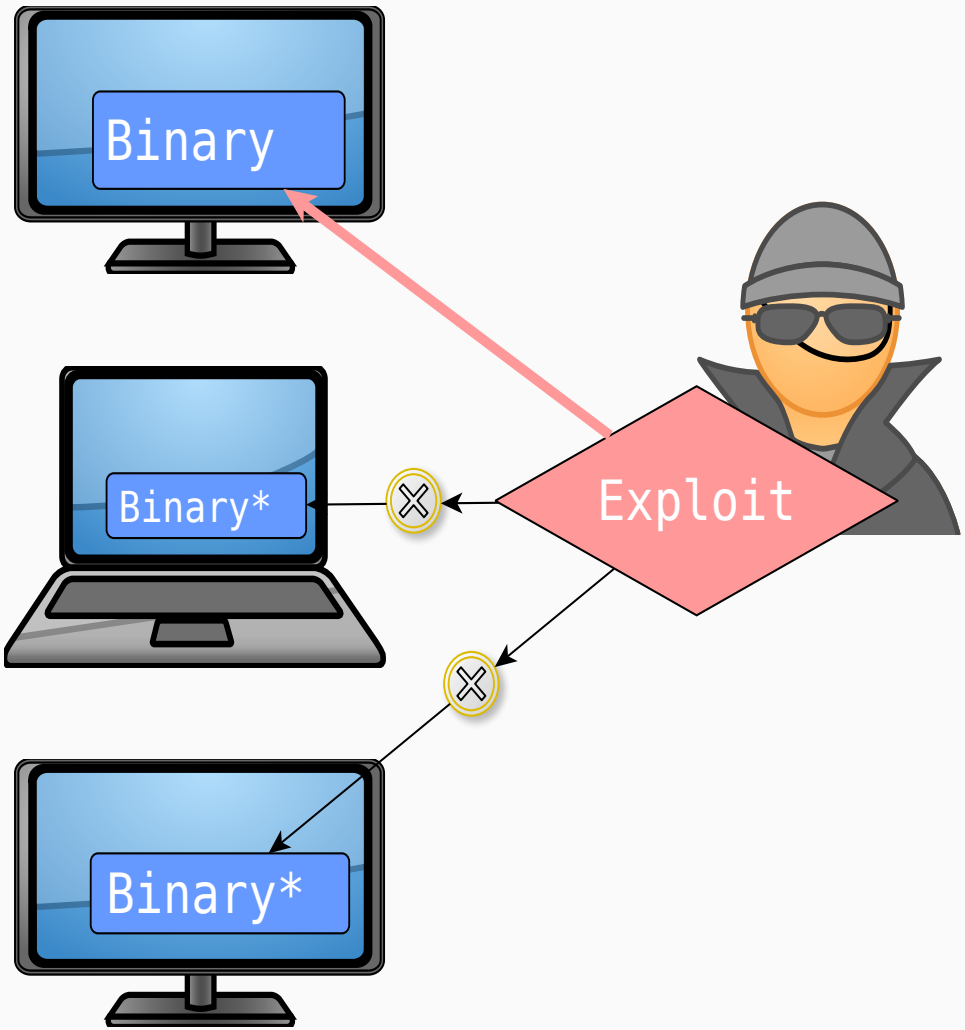
	ASLP	Selfrando	Oxymoron	Runtime
замедление работы, %	~2	~2	~2	100
время запуска	~5 мс	?	0	0
размер файла, %	30	?	2	?
гранулярность	ф-и	ф-и	стр.	?
разделяемость	нет	нет	да	нет

Противодействие эксплуатации уязвимостей

Процедура тестирования

1. Подготовка тестового набора.
2. Поиск и классификация гаджетов.
3. Оценка вероятности гаджета выжить при рандомизации.
4. Генерация rop-цепочек для исходных файлов.
5. Проверка работоспособности сгенерированных rop-цепочек для рандомизированных файлов.

Модель атаки



Тестовый набор файлов

- CentOS 7
- no-PIE ELF
- /usr/bin/*
- /usr/sbin/*

итого 470 файлов

Методика генерации данных для анализа

- Реальная карта памяти процесса получалась модифицированным gcore (gdb).
- Для каждого файла из тестового набора генерировалось по 10 дампов (elf).

Итого 470 * 11 ELF файлов для анализа

Оценка вероятности гаджета выжить при рандомизации

$$\frac{\sum_{j=1}^{m}\left(\frac{\sum_{i=1}^{n_j} k_i^j}{10 n_j}\right)}{m} = 0.054 \quad (1)$$

m - количество файлов,

n_j - количество гаджетов в j файле,

k_i^j - количество файлов, в которых гаджет остался на своем месте.

ROP компилятор

Позволяет:

- искать и классифицировать гаджеты,
- строить из них rop-цепочки.

Примеры для генерации ROP-цепочек

1. foo();
2. foo(1);
3. foo(1, 2);
4. foo(1, 2, 3);
5. system("/bin/sh");

Пример сгенерированных цепочек для sed

foo(1, 2, 3);

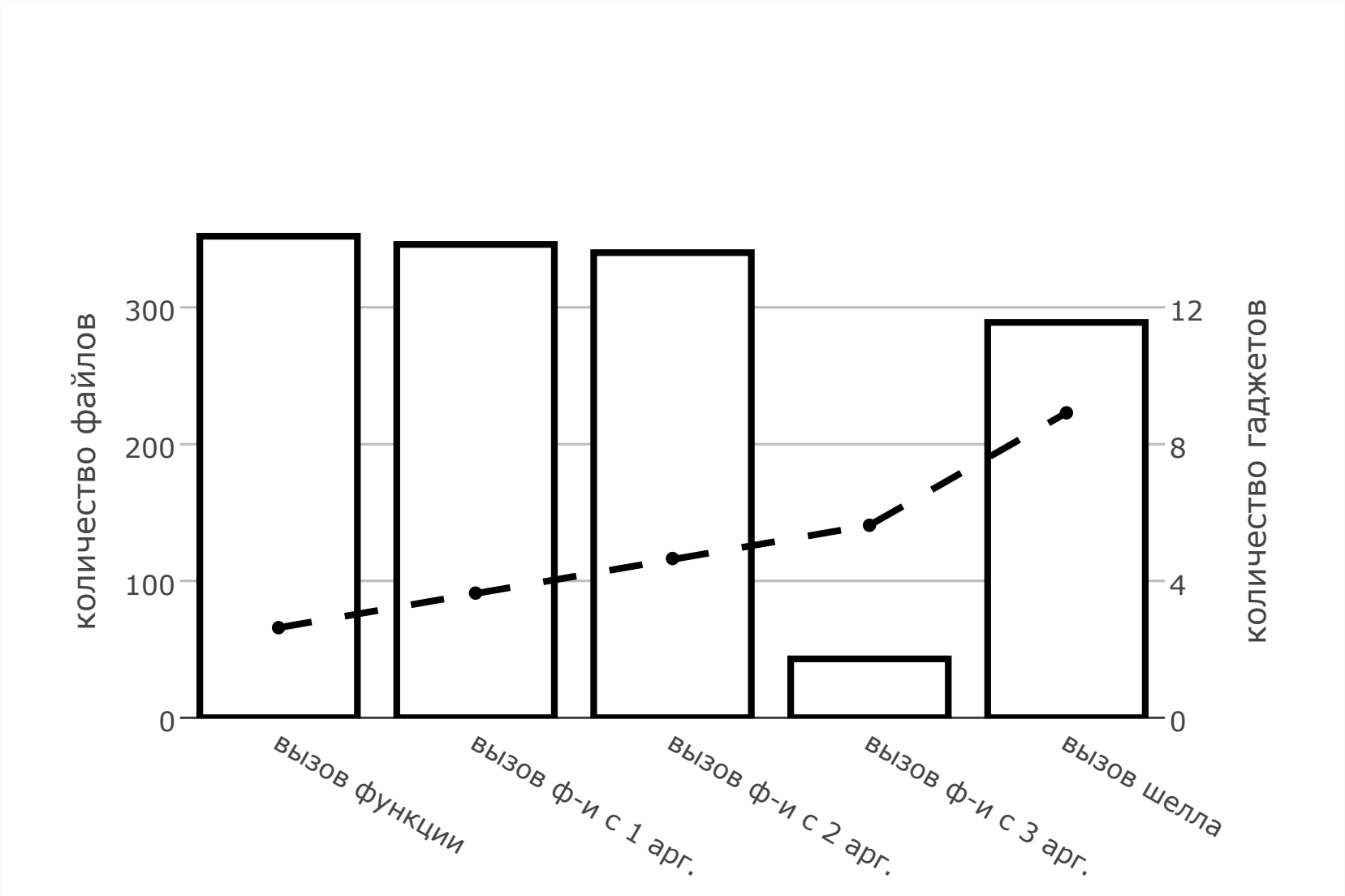
```
0x40b99c -> POP RBX ; RET
0x402e8c -> MOV RAX, RBX ; POP RBX ; RET
0x401de2 -> POP RDX ; RET 0021h
0x40968b -> POP RSI ; RET
0x40bd23 -> POP RDI ; RET
0x4027e7 -> JMP RAX
```

Пример сгенерированных цепочек для sed

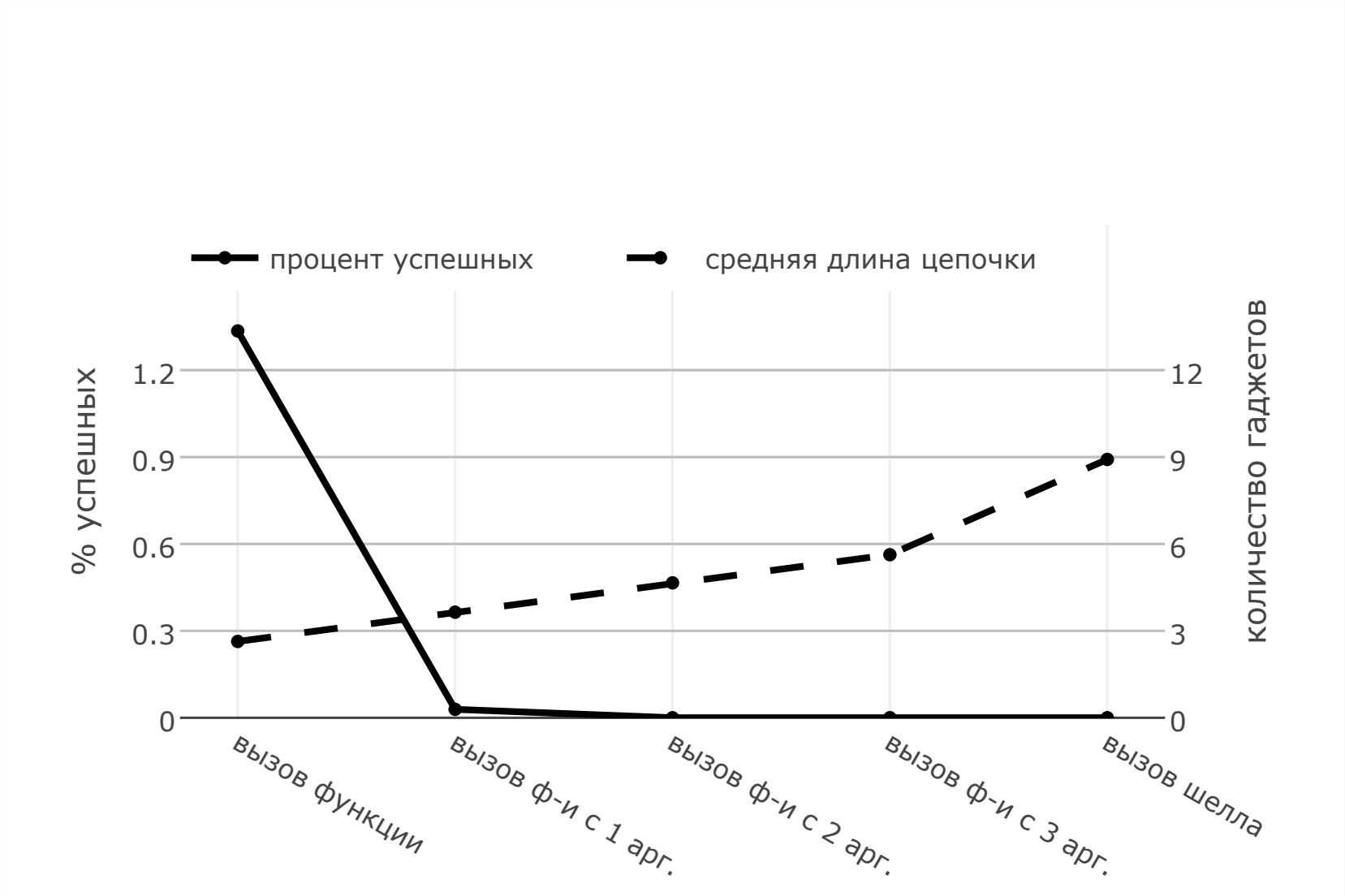
system("/bin/sh");

```
0x401de2 -> POP RDX ; RET 0021h
0x40bd23 -> POP RDI ; RET
0x40ace4 -> MOV QWORD PTR [RDI + 30h], RDX ;
            ADD RSP, 0000000000000008h ; RET
0x40b99c -> POP RBX ; RET
0x402e8c -> MOV RAX, RBX ; POP RBX ; RET
0x40bd23 -> POP RDI ; RET
0x4027e7 -> JMP RAX
```

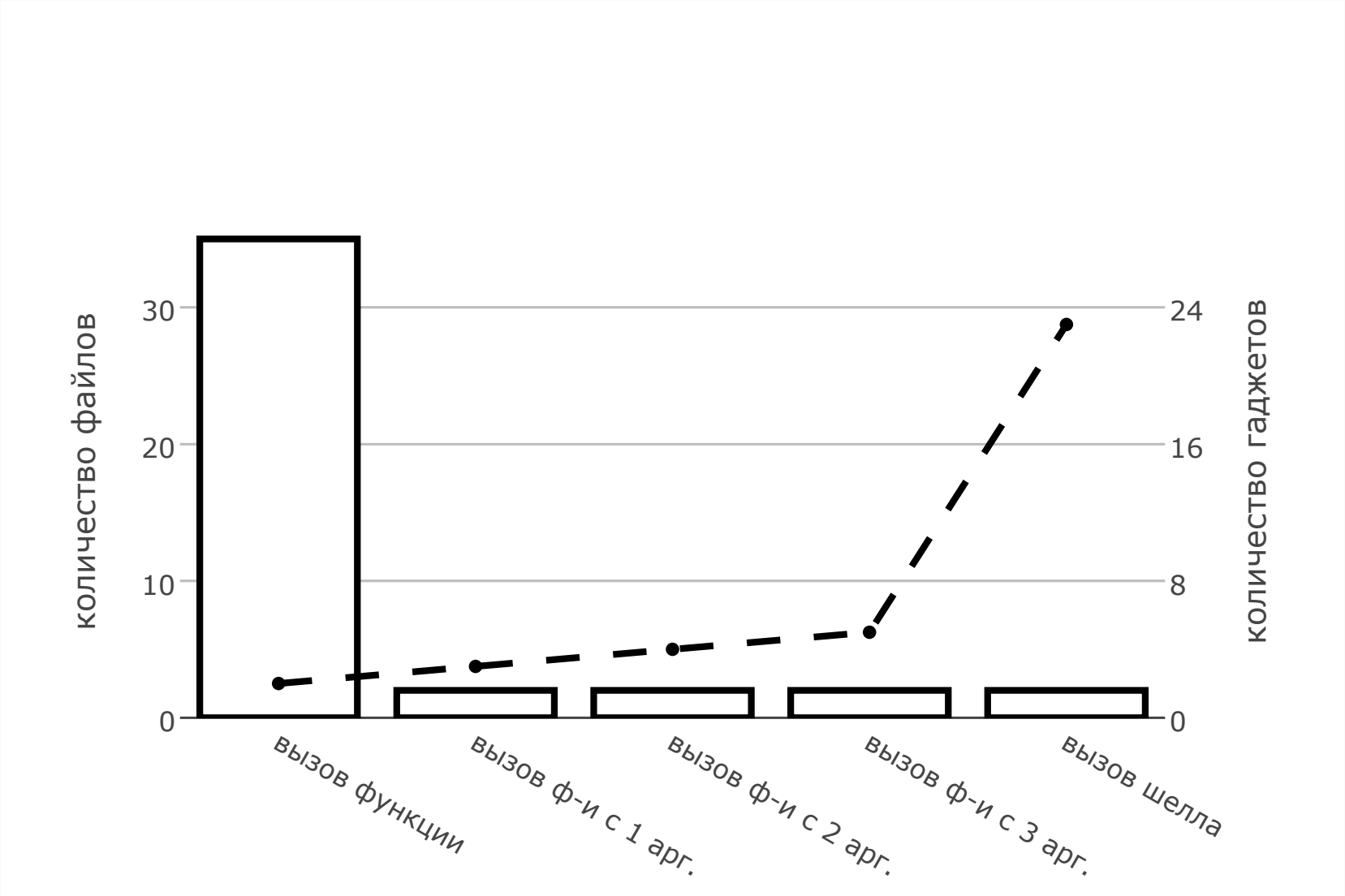
Статистика сгенерированных ROP-цепочек



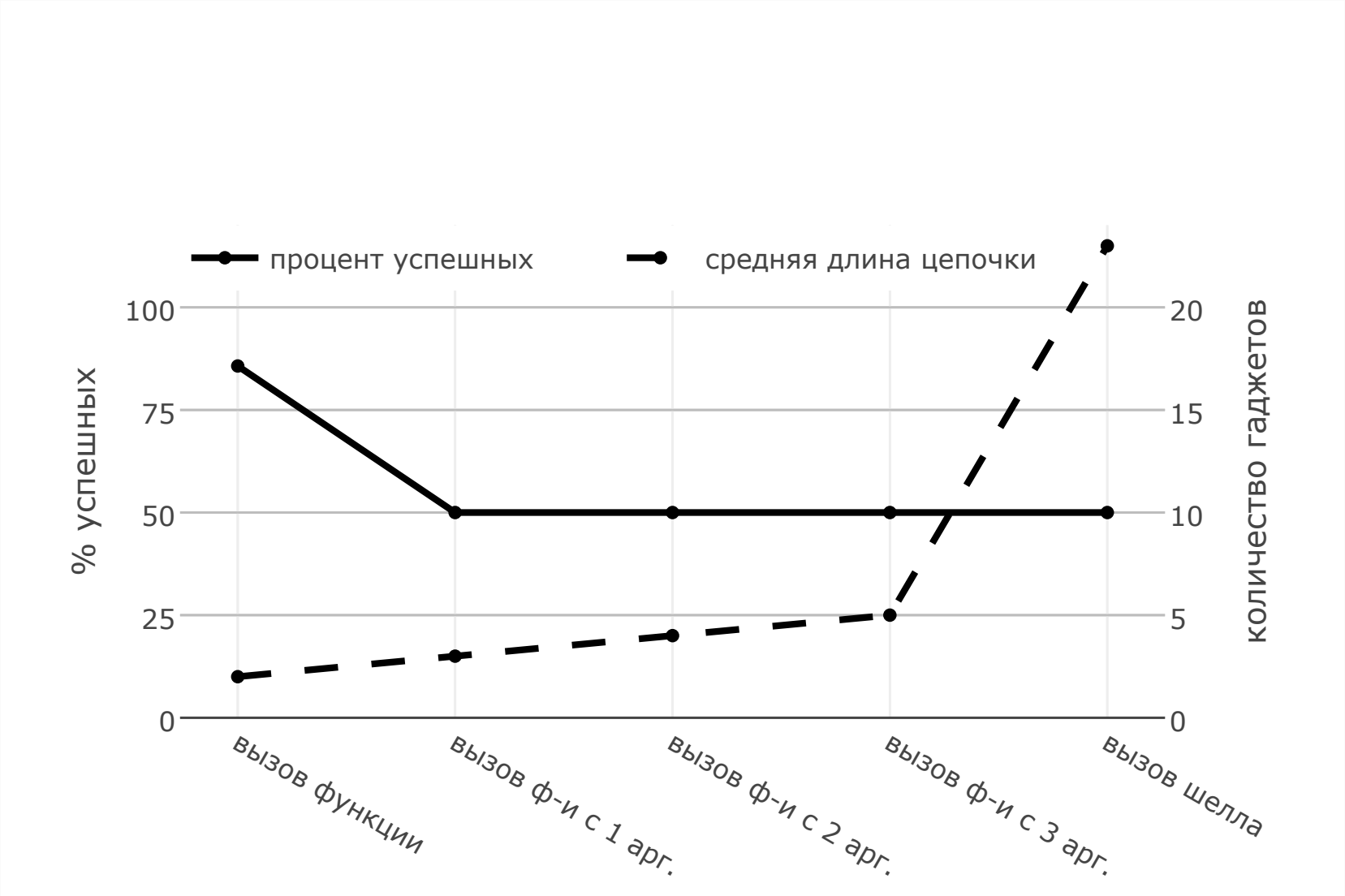
Работоспособность оригинальной ROP-цепочки



Notext статистика сгенерированных rop-цепочек



Notext работоспособность оригинальной rop-цепочки



Заключение

Ограничения

- Затруднена отладка рандомизируемых программ.
- Не поддерживается 32-битные программы и другие архитектуры.
- Не поддерживается гранулярность мельче функции.

Выводы

- Реализованы мелкогранулярная рандомизация адресного пространства при запуске программм с гранулярностью на уровне функций для Linux x86-64.
- Реализация применима в масштабах всей системы.
- Среднее замедление ~ 2 %.
- Произведено тестирование эффективности противодействия эксплуатации методами ROP.

Спасибо за внимание!