

Повышение стойкости программного обеспечения к эксплуатации уязвимостей

30 ноября 2016 г.

Содержание:

Цель и постановка задачи

Уязвимости программного обеспечения

Способы защиты от уязвимостей

Модификация среды исполнения программы

- Executable space protection

- ASLR

- ASLP

Компиляторные методы

- StackGuard

- StackShield

- PointGuard

- G-free

- Return-less technique

Описание предлагаемого метода

Реализованные преобразования

- Добавление локальных переменных

- Случайная перестановка локальных переменных

- Случайная перестановка местами функций

Проверка работоспособности

- Программа с уязвимостью

- Компиляция

- Запуск

Влияние на производительность

Заключение

разработать методы противодействия эксплуатации уязвимостей программного обеспечения, распространяемого через магазины приложений

Уязвимость - недостаток в программе, используя который, можно намеренно нарушить её целостность и вызвать неправильную работу.

Эксплойт - компьютерная программа, использующая уязвимости в программном обеспечении и применяемая для проведения атаки на вычислительную систему.

Программы, содержащие уязвимости, – путь для атак на компьютеры пользователей.

- ▶ проверка исходного кода или бинарного кода людьми и автоматическими анализаторами
- ▶ модификация среды исполнения программы (неисполняемый стек, рандомизация адресного пространства программы)
- ▶ компиляторные методы

Модификация среды исполнения программы

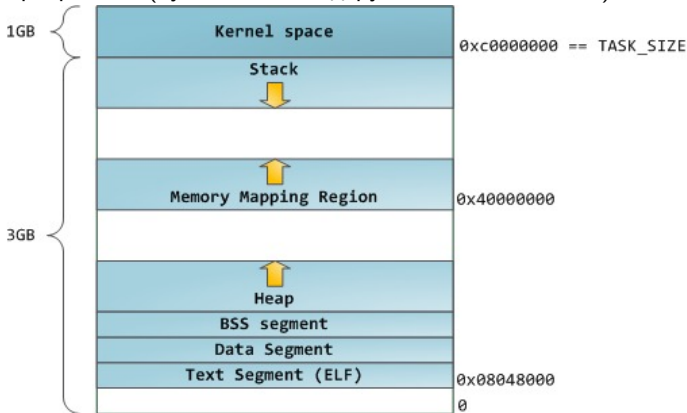
- ▶ Executable space protection
- ▶ ASLR (Address Space Layout Randomization)
- ▶ ASLP (Address Space Layout Permutation)

позволяет задавать привилегии на запись/выполнение отдельных страниц в памяти, т.е. запрет на исполнение стека/кучи и запрет записи в секцию кода программы.

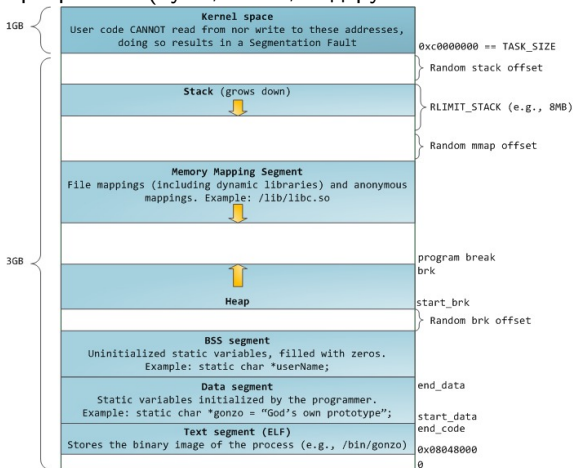
Работает при соблюдении условий:

- ▶ аппаратная поддержка NX-bit/XD-bit у AMD/Intel
- ▶ поддержка со стороны ОС (linux kernel 2.3.23, Windows XP SP2)
- ▶ использование PAE или архитектуры x86-64

случайно изменяет расположение в адресном пространстве секций программы (кучи, стека, подгружаемых библиотек)



случайно изменяет расположение в адресном пространстве секций программы (кучи, стека, подгружаемых библиотек)



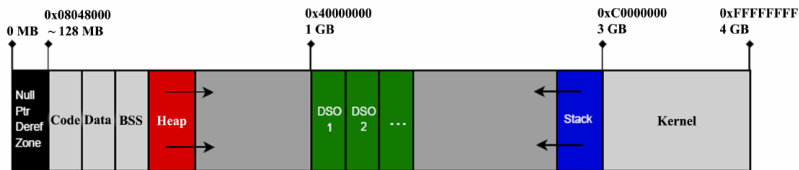
Ограничения:

- ▶ недостаточная рандомизация адресного пространства программы (особенно на 32-битных архитектурах)
- ▶ сохранение относительной последовательности секций программы
- ▶ каждая секция внутри остается неизменной

случайно меняет размещение в памяти программы при каждом ее запуске.

Используется:

- ▶ крупнозернистая перестановка - меняет местами все секции программы без сохранения их относительной последовательности

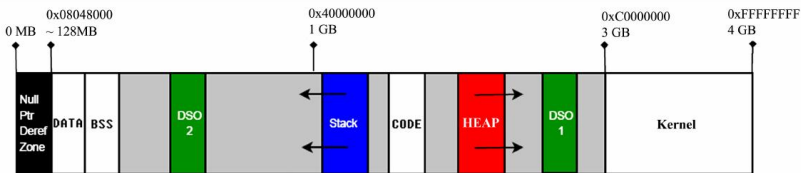


- ▶ мелкозернистая перестановка - случайно меняет местами функции, статические и глобальные переменные внутри секций кода и данных

случайно меняет размещение в памяти программы при каждом ее запуске.

Используется:

- ▶ крупнозернистая перестановка - меняет местами все секции программы без сохранения их относительной последовательности



- ▶ мелкозернистая перестановка - случайно меняет местами функции, статические и глобальные переменные внутри секций кода и данных

представляет собой утилиту, меняющую бинарный файл каждый раз при загрузке на исполнение.

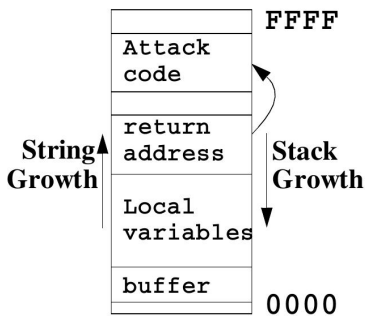
Ограничения:

- ▶ для корректной работы требуются relocation data от компилятора
- ▶ не производится мелкозернистая перестановка внутри секции стека

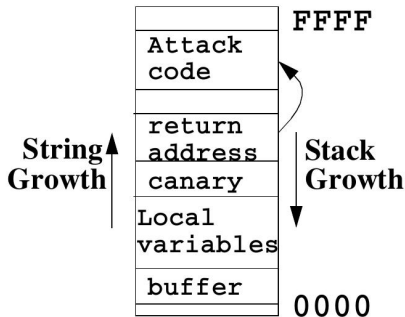
Компиляторные методы защиты от использования уязвимостей

- ▶ StackGuard
- ▶ StackShield
- ▶ PointGuard
- ▶ G-free
- ▶ Return-less binary

Пролог функций изменяется таким образом, чтобы вставлять на стек после адреса возврата проверочное слово (canary). В эпилог функции добавляется проверка того, изменилось ли это слово.



стек



защищенный стек

Падение производительности при добавлении StackGuard в каждую функцию примерно 10%.

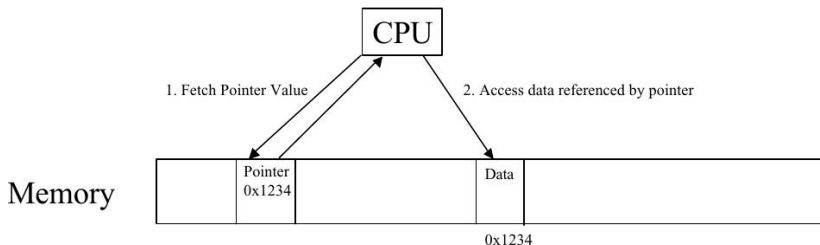
присутствует в компиляторах

- ▶ Microsoft VisualStudio: /GS флаг включен по умолчанию
- ▶ GCC: -fstack-protector -fstack-protector-all -fstack-protector-strong
- ▶ Clang/LLVM: -fstack-protector

- ▶ представляет собой утилиту, которая меняет исходный ассемблерный файл
- ▶ адреса возврата копируются в теневой стек в прологе функции
- ▶ При выходе из функции :
 1. используются только адреса возврата из теневого стека
 2. проверяется соответствие, в случае несовпадения используется адрес возврата из теневого стека
 3. проверяется соответствие, в случае несовпадения прерывается выполнение программы

защищает указатели программы:

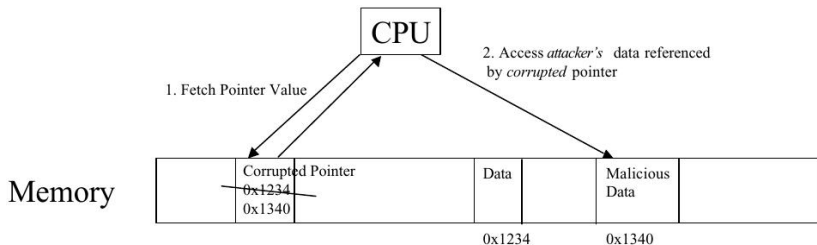
- ▶ в памяти хранятся только зашифрованные значения указателя
- ▶ расшифрованное значение указателя хранится только на регистре



разыменование обычного указателя

защищает указатели программы:

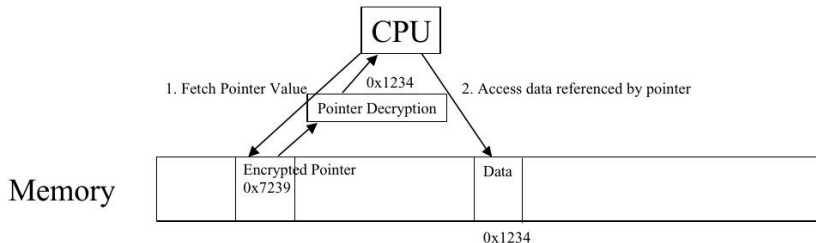
- ▶ в памяти хранятся только зашифрованные значения указателя
- ▶ расшифрованное значение указателя хранится только на регистре



разыменование обычного указателя под атакой

защищает указатели программы:

- ▶ в памяти хранятся только зашифрованные значения указателя
- ▶ расшифрованное значение указателя хранится только на регистре

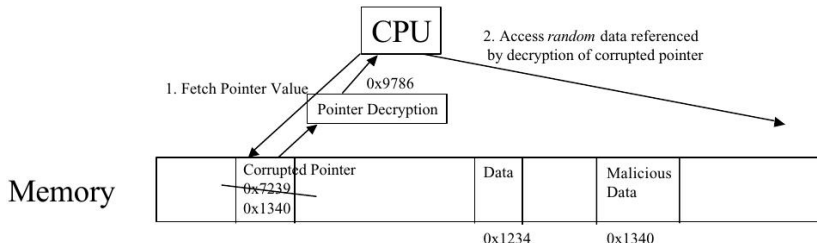


разыменование защищенного указателя

Падение производительность 0-20%.

защищает указатели программы:

- ▶ в памяти хранятся только зашифрованные значения указателя
- ▶ расшифрованное значение указателя хранится только на регистре



разыменование защищенного указателя под атакой

Падение производительность 0-20%.

ROP (return-oriented programming)

Возвратно-ориентированное программирование

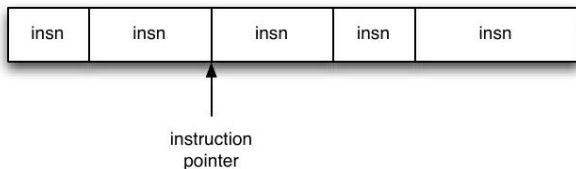
- ▶ не требует инъекции вредоносного кода
- ▶ использует последовательность гаджетов

Гаджет - небольшой кусок машинного кода,
заканчивающийся инструкцией возврата

- ▶ возможно построить Тьюринг полный набор гаджетов

ROP (return-oriented programming)

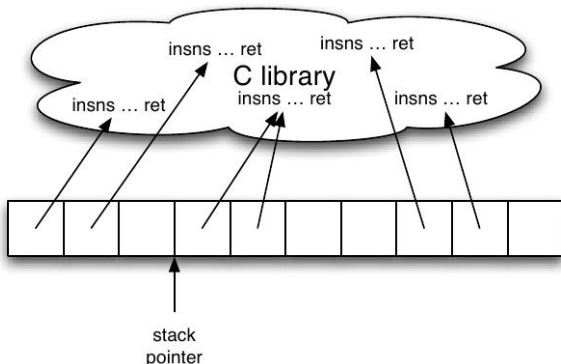
Возвратно-ориентированное программирование



последовательность инструкций обычной программы

ROP (return-oriented programming)

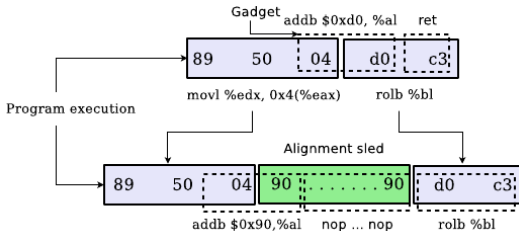
Возвратно-ориентированное программирование



последовательность инструкций ROP программы

представляет собой препроцессор для ассемблера и утилиту бинарного анализа и использует методы:

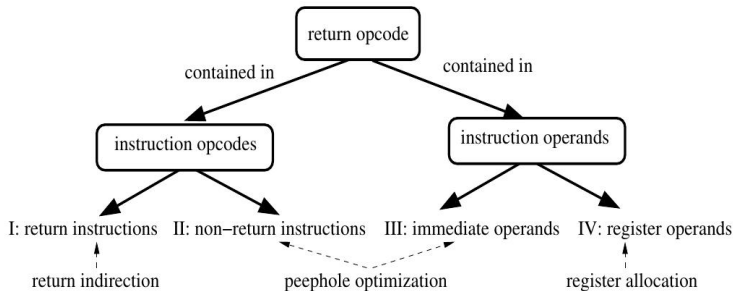
- ▶ вставка блоков выравнивания для разрыва гаджета



- ▶ перераспределение регистров
- ▶ реерhole-оптимизации
- ▶ шифрование адреса возврата

Размер glibc увеличился на 30%. Падение производительности оценивается 3-5%.

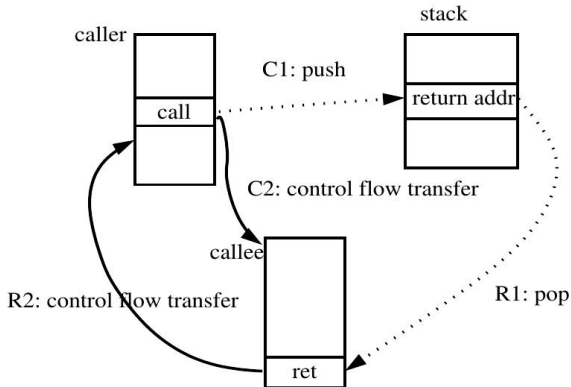
полное исключение всех инструкций возврата из машинного кода



Return-less technique

Непрямой возврат

На стеке вместо адреса возврата функции сохраняется его индекс в специальной таблице

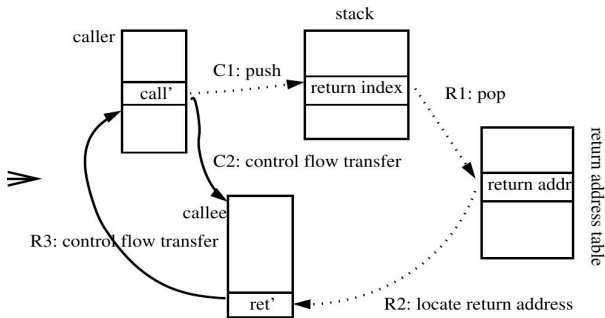


обычная передача потока управления

Return-less technique

Непрямой возврат

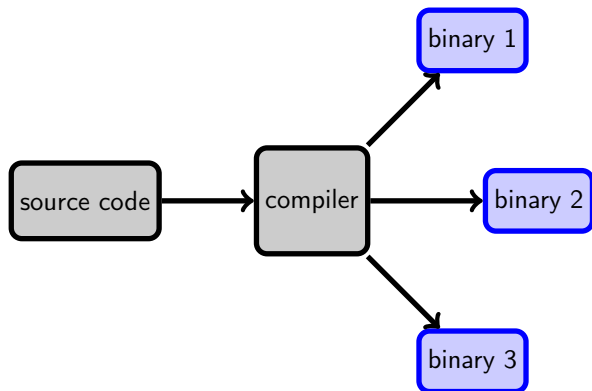
На стеке вместо адреса возврата функции сохраняется его индекс в специальной таблице



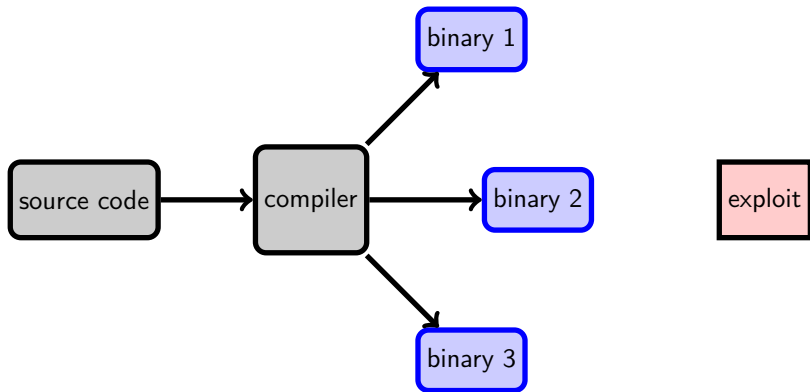
непрямая передача потока управления

- ▶ инструмент реализован на инфраструктуре LLVM.
- ▶ авторами получен образ ядра FreeBSD/x86-amd64 8.0 абсолютно без инструкций возврата (около 18000).
- ▶ падение производительности 5-15%.
- ▶ размер образа ядра увеличился на 10%.

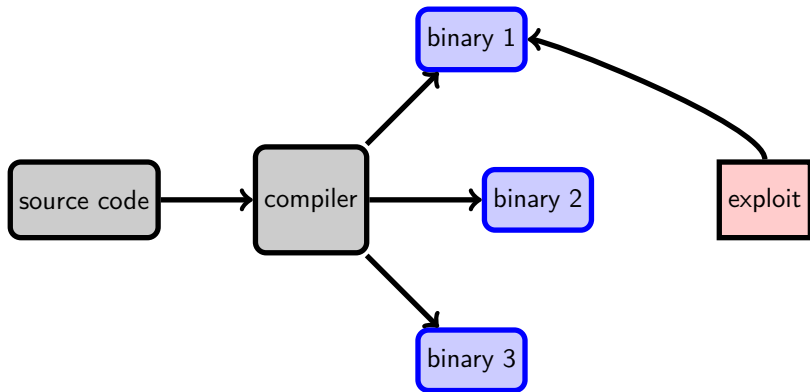
Получение различных исполняемых файлов



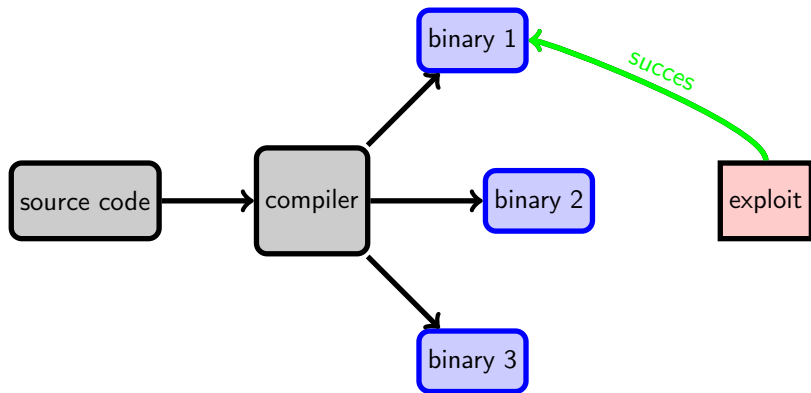
Получение различных исполняемых файлов



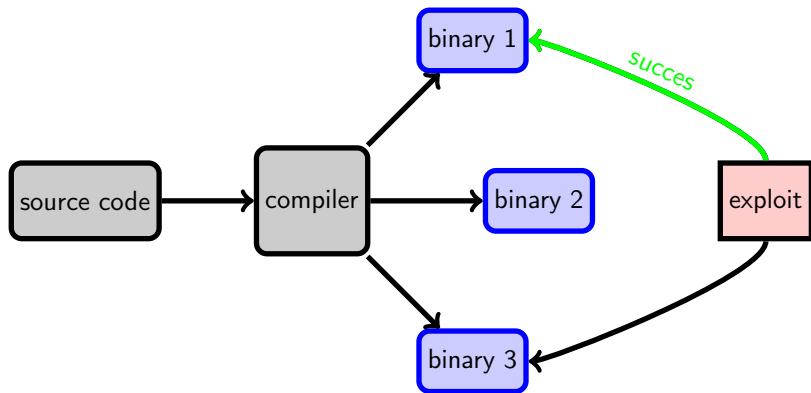
Получение различных исполняемых файлов



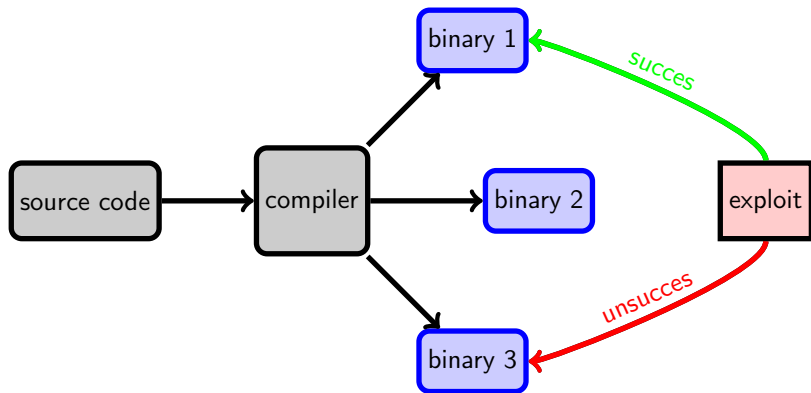
Получение различных исполняемых файлов



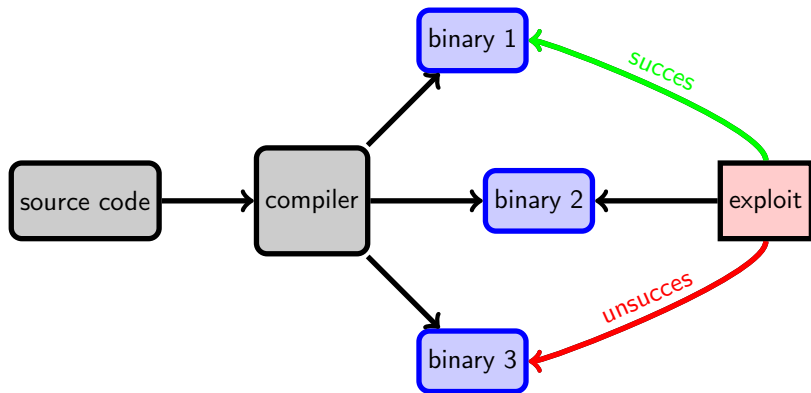
Получение различных исполняемых файлов



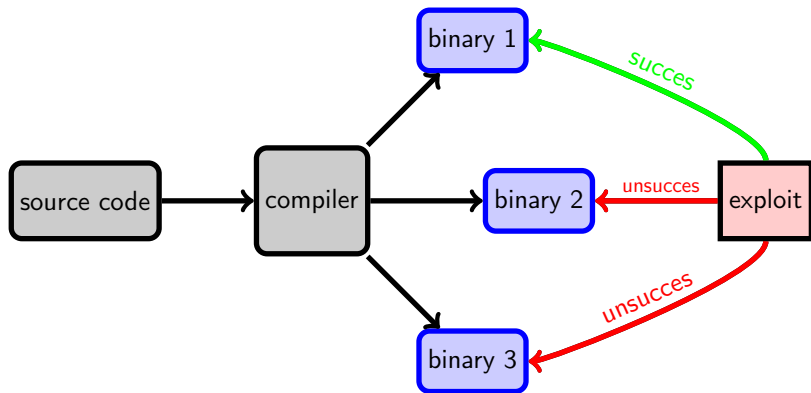
Получение различных исполняемых файлов



Получение различных исполняемых файлов



Получение различных исполняемых файлов



- ▶ добавление локальных переменных
- ▶ случайная перестановка локальных переменных на стеке
- ▶ случайная перестановка местами функций

До преобразования

```
define i32 @f() {  
  %s = alloca i16  
  %f = alloca float  
  %a = alloca [10 x i32]  
  %i = alloca i32  
  ...  
}
```

После преобразования

```
define i32 @f() {  
  %1 = alloca i32  
  %2 = alloca i32  
  %3 = alloca i32  
  %4 = alloca i32  
  %s = alloca i16  
  %f = alloca float  
  %a = alloca [10 x i32]  
  %i = alloca i32  
  ...  
}
```

Случайная перестановка локальных переменных на стеке

До преобразования

```
define i32 @f() {  
    %s = alloca i16  
    %f = alloca float  
    %i = alloca i32  
    %a = alloca [10 x i32]  
    ...  
}
```

После преобразования

```
define i32 @f() {  
    %s = alloca i16  
    %i = alloca i32  
    %f = alloca float  
    %a = alloca [10 x i32]  
    ...  
}
```


До преобразования

```
define i32 @f() {  
    ...  
}  
define i32 @g() {  
    ...  
}  
define i32 @h() {  
    ...  
}  
define i32 @main() {  
    ...  
    %f = call i32 @f()  
    %g = call i32 @g()  
    %h = call i32 @h()  
    ...  
}
```

После преобразования

```
define i32 @h() {  
    ...  
}  
define i32 @g() {  
    ...  
}  
define i32 @main() {  
    ...  
    %f = call i32 @f()  
    %g = call i32 @g()  
    %h = call i32 @h()  
    ...  
}  
define i32 @f() {  
    ...  
}
```

Программа с уязвимостью переполнения буфера

```
// Module = test.c
#include <stdio.h>
#include <stdlib.h>

void foo() {
    printf ("foo\n");
    return;
}

void boo (char *s) {
    char name[6];
    strcpy (name, s);
    return;
}

int main (int argc, char **argv) {
    boo (argv[1]);
    return 0;
}
```

обычной версии программы

```
clang test.c -o binary
```

защищенной программы

```
clang test.c -allocs-re -functions-re \  
            -add-allocs=10 -o permuted_binary
```

обычной версии программы

```
$ ./binary aaaaaaaaaaaaaaaaaaaaaa  
foo  
Segmentation fault (core dumped)
```

защищенной программы

```
$ ./permuted_binary aaaaaaaaaaaaaaaaaaaaaa  
Segmentation fault (core dumped)
```

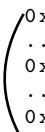
Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0xffffe576 0x00007fff 0xffffe7b1 0x00007fff
0x7fffffff400: 0xffffe430 0x00007fff 0x00400583 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```




Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0xff61e576 0x00007fff 0xffffe7b1 0x00007fff
0x7fffffff400: 0xffffe430 0x00007fff 0x00400583 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```



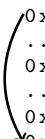
Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0x6161e576 0x00007f61 0xffffe7b1 0x00007fff
0x7fffffff400: 0xffffe430 0x00007fff 0x00400583 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```




Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0x6161e576 0x00006161 0xffffe7b1 0x00007fff
0x7fffffff400: 0xffffe430 0x00007fff 0x00400583 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```



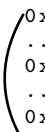
Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0x6161e576 0x61616161 0x61616161 0x61616161
0x7fffffff400: 0x61616161 0x00006161 0x00400583 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```




Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0x6161e576 0x61616161 0x61616161 0x61616161
0x7fffffff400: 0x61616161 0x00616161 0x00400583 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```



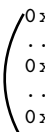
Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0x6161e576 0x61616161 0x61616161 0x61616161
0x7fffffff400: 0x61616161 0x61616161 0x00400583 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```




Запуск обычной версии программы

Стек функции boo

```
0x7fffffff3e0: 0x00000000 0x00000000 0x004003c3 0x00000000
0x7fffffff3f0: 0x6161e576 0x61616161 0x61616161 0x61616161
0x7fffffff400: 0x61616161 0x61616161 0x00400500 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400530 <boo>:
...
0x400547:      callq  4003e0 <strcpy@plt>
...
0x40055c:      retq
...
0x400560 <main>:
...
0x40057e:      callq  400530 <boo>
0x400583:      mov    $0x0,%ecx
...
```



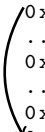
Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x00400650 0x00000000
0x7fffffff3b0: 0xffffe400 0x00007fff 0x0040063f 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...
```



Запуск защищенной версии программы

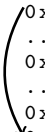
Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x00610650 0x00000000
0x7fffffff3b0: 0xffffe400 0x00007fff 0x0040063f 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...

```



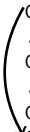
Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x61610650 0x00000000
0x7fffffff3b0: 0xffffe400 0x00007fff 0x0040063f 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...
```



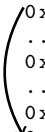
Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x61610650 0x00006161
0x7fffffff3b0: 0xffffe400 0x00007fff 0x0040063f 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...
```



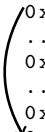
Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x61610650 0x61616161
0x7fffffff3b0: 0x61616161 0x61616161 0x0040063f 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...
```



Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x61610650 0x61616161
0x7fffffff3b0: 0x61616161 0x61616161 0x00400661 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...

```

Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x61610650 0x61616161
0x7fffffff3b0: 0x61616161 0x61616161 0x00406161 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...

```

Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x61610650 0x61616161
0x7fffffff3b0: 0x61616161 0x61616161 0x00616161 0x00000000
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...

```

Запуск защищенной версии программы

Стек функции boo

```
0x7fffffff370: 0x00000000 0x00000000 0xf7abb320 0x00007fff
0x7fffffff380: 0x00000000 0x00000000 0x00000076 0x00000000
0x7fffffff390: 0x00000009 0x00000000 0x000000c2 0x00000000
0x7fffffff3a0: 0xffffe781 0x00007fff 0x61610650 0x61616161
0x7fffffff3b0: 0x61616161 0x61616161 0x61616161 0x61616161
```

Секция кода программы

```
0x400500 <foo>:
...
0x400570 <boo>:
...
0x4005c3:      callq  4003e0 <strcpy@plt>
...
0x4005d8:      retq
...
0x4005e0 <main>:
...
0x40063a:      callq  400570 <boo>
0x40063f:      mov    $0x0,%ecx
...

```

Реализованные компиляторные преобразования негативно сказываются на производительности получаемых программ. На приложении SQLite было зафиксировано падение производительности на 30% при добавлении в каждую функцию по 30 локальных переменных типа `int`.

Предлагаемый подход

- ▶ позволяет повысить устойчивость к эксплуатации уязвимостей за счет генерации различных версий исполняемого файла программы
- ▶ может успешно сочетаться с другими средствами защиты

Дальнейшие пути развития:

- ▶ реализация защиты от ROP атак

Спасибо за внимание