Санкт-Петербургский государственный университет

Кафедра системного программирования

Обфускация самомодифицирующимся кодом

Юлия Владиславовна Чернобровкина, группа 22.Б08-мм

Научный руководитель: К.К. Смирнов, ст. преподаватель кафедры ИАС
Консультант: П.А. Бабанов, старший разработчик ПО I категории ООО «Софтком»

Санкт-Петербург
2025

Введение

- Обфускация как метод защиты ПО
- Самомодифицирующийся код
- Обфускатор компании «Софтком»
- Требования компании
  - Целевая архитектура x86-64
  - Интеграция кода в фреймворк LLVM

Постановка задачи

Целью работы является исследование возможности реализации метода самомодификации кода на этапе компиляции в инфраструктуре LLVM для обфускатора компании «Софтком»

Задачи:

- провести обзор существующих решений и используемых технологий
- реализовать прототип метода самомодификации кода на основе инфраструктуры LLVM
- выявить ключевые ограничения, возникающие при реализации самомодификации на этапе компиляции
- провести экспериментальное исследование полученного прототипа

Существующие решения

- Tigress
  - Есть методы самомодификации
  - Инструмент доступен пользователям
  - Исходный код закрыт
  - Целевой язык C
- Themida
  - Есть метод самомодификации
  - Исходный код закрыт
  - Инструмент коммерческий
- Obsidium
  - Есть методы самомодификации
  - Исходный код закрыт
  - Инструмент коммерческий
  - Поддержка только Windows

Фреймворк LLVM

Разработка обфускатора ведется в инфраструктуре LLVM.

- Документированное API для разработки фронтэндов, бекэндов, проходов
- Классы для архитектур, модулей, функций и так далее

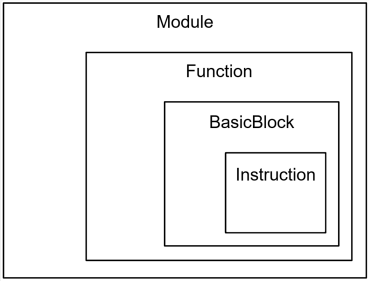

Рис.: Классы LLVM API. Изображение взято из источника `https://sungsoo.github.io/2016/04/24/introduction-to-llvm.html`(дата обращения: 24 марта 2025 г.).

Метод самомодификации

- Динамическое шифрование/дешифрование бинарного кода
- Метод шифрования — прибавление константы
- Интеграция с системой разметки исходного кода

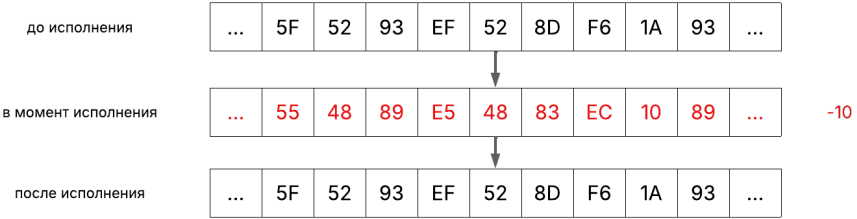

Рис.: Байты кода обфусцированной функции до исполнения, в момент и после исполнения.

Реализация

Для обфускации в архитектуре x86-64 — отдельный таргет

Изменения на разных уровнях кодогенерации в LLVM:

- LLVM IR
  - Добавление функции шифратора
  - Механизм проходов LLVM Passes
- MachineInstruction (ассемблер)
  - Извлечение метаданных
  - Обрамляющие шифрующие/дешифрующие блоки
  - Замена инструкций возврата на безусловный переход на выходной блок
  - Помещение функции в секцию .data и добавление секции прав на исполнение (X)
- MCInst (машинные инструкции)
  - Прибавление константы к байтам кода

Рис.: Схема трансформаций инструкций при компиляции в LLVM.

Добавление обрамляющих блоков

исходная функция

```
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq  %rsp, %rbp
.cfi_def_cfa_register %rbp
subq  $16, %rsp
movl  %edi, -4(%rbp)
movq  %rsi, -16(%rbp)
movq  -16(%rbp), %rdi
callq puts@PLT
addq  $16, %rsp
popq  %rbp
.cfi_def_cfa %rsp, 8
retq
```

модифицированная функция

дешифрующий блок

```
pushq %rdi
pushq %rsi
movabsq $func_begin_foo, %rdi
movabsq $func_end_foo, %rsi
subq  %rdi, %rsi
movb  $-10, %dl
callq encrypt_func
popq  %rsi
popq  %rdi
```

метка начала

```
func_begin_foo:
```

```
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq  %rsp, %rbp
.cfi_def_cfa_register %rbp
subq  $16, %rsp
movl  %edi, -4(%rbp)
movq  %rsi, -16(%rbp)
movq  -16(%rbp), %rdi
callq puts@PLT
addq  $16, %rsp
popq  %rbp
.cfi_def_cfa %rsp, 8
jmp   .LBB1_2
```

метка конца

```
.LBB1_2:
func_end_foo:
```

шифрующий блок

```
pushq %rdi
pushq %rsi
movabsq $func_begin_foo, %rdi
movabsq $func_end_foo, %rsi
subq  %rdi, %rsi
movb  $10, %dl
callq encrypt_func
popq  %rsi
popq  %rdi
retq
```

Ограничения метода

- Ограничения на структуру функци:
  - Отсутствие циклов и ветвлений
  - Отсутствие вызовов других функций модуля
  - Отсутствие обращений к глобальным переменным
- Причины ограничений:
  - Шифрование выполняется на этапе компиляции
  - Релокации обрабатываются компоновщиком
  - Модификация зашифрованных адресов компоновщиком создает некорректный код

Экспериментальное исследование

- Фиксация ограничений текущей реализаций:
  - Тесты LLVM lit
- Постановка эксперимента:
  - Тестовая программа на языке C
  - ПО: LLVM 19.0.0., clang 19.0.0
  - Дизассемблеры:
    - Ghidra version 11.2.1
    - IDA Free 9.0

Экспериментальное исследование

Listing 1: Тестовая программа, вызывающая printf

```
/*?
{
    "type": "method",
    "obfuscate":
    {
        "add_const": {
            "type": "int",
            "value": 10
        }
    }
}
?*/
void print_str_num(int a, const char *str){
    printf("Number: %d\nString: %s\n", a, str);
}
```

Декомпиляция обфусцированного кода (Ghidra, часть 1)

```
                    undefined print_str_num()
     undefined      ⚠ <UNASSIGNED>   <RETURN>
                    print_str_num                                XREF[4]:     Entry Point(*), main:00101168(c),
                                                                              0010207c, 00102118(*)
00104030 50              PUSH       RAX
00104031 57              PUSH       RDI
00104032 56              PUSH       RSI
00104033 48 bf 54        MOV        RDI,0x104054
         40 10 00
         00 00 00 00
0010403d 48 be 82        MOV        RSI,func_end_print_str_num
         40 10 00
         00 00 00 00
00104047 48 29 fe        SUB        RSI,RDI
0010404a b2 f6           MOV        DL,0xf6
0010404c e8 2f d1        CALL       decrypt_func                          undefined decrypt_func()
         ff ff
00104051 5e              POP        RSI
00104052 5f              POP        RDI
00104053 58              POP        RAX
```

Рис.: Начало функции print_str_num (декомпиляция Ghidra)

Декомпиляция обфусцированного кода (Ghidra, часть 2)

```
                    func_begin_print_str_num
00104054 5f         POP     RDI
00104055 52         PUSH    RDX
00104056 93         XCHG    EAX,EBX
00104057 ef         OUT     DX,EAX
00104058 52         PUSH    RDX
00104059 8d         ??      8Dh
0010405a f6         ??      F6h
0010405b 1a         ??      1Ah
0010405c 93         ??      93h
0010405d 87         ??      87h
0010405e 06         ??      06h
0010405f 52         ??      52h    R
00104060 93         ??      93h
00104061 7f         ??      7Fh    
00104062 fa         ??      FAh
00104063 95         ??      95h
00104064 7f         ??      7Fh    
00104065 06         ??      06h
00104066 52         ??      52h    R
00104067 95         ??      95h
00104068 5f         ??      5Fh    _
00104069 fa         ??      FAh
0010406a 52         ??      52h    R
0010406b 97         ??      97h
0010406c 47         ??      47h    G
0010406d 9d         ??      9Dh
0010406e e9         ??      E9h
0010406f 09         ??      09h
00104070 09         ??      09h
00104071 ba         ??      BAh
00104072 0a         ??      0Ah
00104073 f2         ??      F2h
```

Рис.: Середина функции print_str_num (декомпиляция Ghidra)

Декомпиляция обфусцированного кода (Ghidra, часть 3)

```
ting: obf_program
00104076 09          ??        09h
00104077 09          ??        09h
00104078 52          ??        52h    R
00104079 8d          ??        8Dh
0010407a ce          ??        CEh
0010407b 1a          ??        1Ah
0010407c 67          ??        67h    g
0010407d f3          ??        F3h
0010407e 0a          ??        0Ah
0010407f 0a          ??        0Ah
00104080 0a          ??        0Ah
00104081 0a          ??        0Ah

                 func_end_print_str_num                    XREF[2]:     print_str_num:0010403d(*),
                                                                        0010408f(*)
00104082 50          PUSH      RAX
00104083 57          PUSH      RDI
00104084 56          PUSH      RSI
00104085 48 bf 54    MOV       RDI,0x104054
         40 10 00
         00 00 00 00
0010408f 48 be 82    MOV       RSI,func_end_print_str_num
         40 10 00
         00 00 00 00
00104099 48 29 fe    SUB       RSI,RDI
0010409c b2 0a       MOV       DL,0xa
0010409e e8 dd d0    CALL      decrypt_func                        undefined decrypt_func()
         ff ff
001040a3 5e          POP       RSI
001040a4 5f          POP       RDI
001040a5 58          POP       RAX
001040a6 c3          RET
                 //
```

Рис.: Завершение функции print_str_num (декомпиляция Ghidra)

Экспериментальное исследование. Декомпиляция обфусцированного кода инструментом IDA Free

```
.data:0000000000004030                 public print_str_num
.data:0000000000004030 print_str_num:                          ; CODE XREF: main+28↑p
.data:0000000000004030 ; __unwind {
.data:0000000000004030                 push    rax
.data:0000000000004031                 push    rdi
.data:0000000000004032                 push    rsi
.data:0000000000004033                 mov     rdi, offset func_begin_print_str_num
.data:000000000000403D                 mov     rsi, offset func_end_print_str_num
.data:0000000000004047                 sub     rsi, rdi
.data:000000000000404A                 mov     dl, 0F6h
.data:000000000000404C                 call    decrypt_func
.data:0000000000004051                 pop     rsi
.data:0000000000004052                 pop     rdi
.data:0000000000004053                 pop     rax
.data:0000000000004054
.data:0000000000004054 func_begin_print_str_num:               ; DATA XREF: .data:000000000000403D↑o
.data:0000000000004054                                         ; func_end_print_str_num+3↓o
.data:0000000000004054                 pop     rdi
.data:0000000000004055                 push    rdx
.data:0000000000004056                 xchg    eax, ebx
.data:0000000000004057                 out     dx, eax
.data:0000000000004058                 push    rdx
.data:0000000000004058 ; ---------------------------------------------------------------------------
.data:0000000000004059                 db 8Dh, 0F6h, 1Ah, 93h, 87h, 6, 52h
.data:000000000000405D                 dq 9552067F955A7F93h, 9E99D479752FA5Fh, 909D9C2F20ABA09h
.data:0000000000004078                 dq 0A0AF3671ACE8D52h
.data:0000000000004080 ; ---------------------------------------------------------------------------
.data:0000000000004080                 or      cl, [rdx]
.data:0000000000004082
.data:0000000000004082 ; =============== S U B R O U T I N E =======================================
.data:0000000000004082
.data:0000000000004082
.data:0000000000004082 func_end_print_str_num proc near        ; DATA XREF: .data:000000000000403D↑o
.data:0000000000004082                                         ; func_end_print_str_num+D↓o
.data:0000000000004082                 push    rax
.data:0000000000004083                 push    rdi
.data:0000000000004084                 push    rsi
.data:0000000000004085                 mov     rdi, offset func_begin_print_str_num
.data:000000000000408F                 mov     rsi, offset func_end_print_str_num
.data:0000000000004099                 sub     rsi, rdi
.data:000000000000409C                 mov     dl, 0Ah
.data:000000000000409E                 call    decrypt_func
.data:00000000000040A3                 pop     rsi
.data:00000000000040A4                 pop     rdi
.data:00000000000040A5                 pop     rax
.data:00000000000040A6                 retn
.data:00000000000040A6 ; } // starts at 4030
```

Рис.: Восстановленная дизассемблером IDA Free обфусцированная функция print_str_num.

Результаты

- проведен обзор существующих решений и используемых технологий
- реализован прототип метода самомодификации кода на основе инфраструктуры LLVM
- выявлены ключевые ограничения, возникающие при реализации самомодификации на этапе компиляции
- проведено экспериментальное исследование полученного прототипа с помощью инструментов статического анализа

Исходный код проекта закрыт