SAFEcode 2024

Почему kernel так сложно защитить

и что можно сделать

Анна Мелехова,
руководитель группы
разработки защитных
решений безопасной
платформы,
«Лаборатория Касперского»

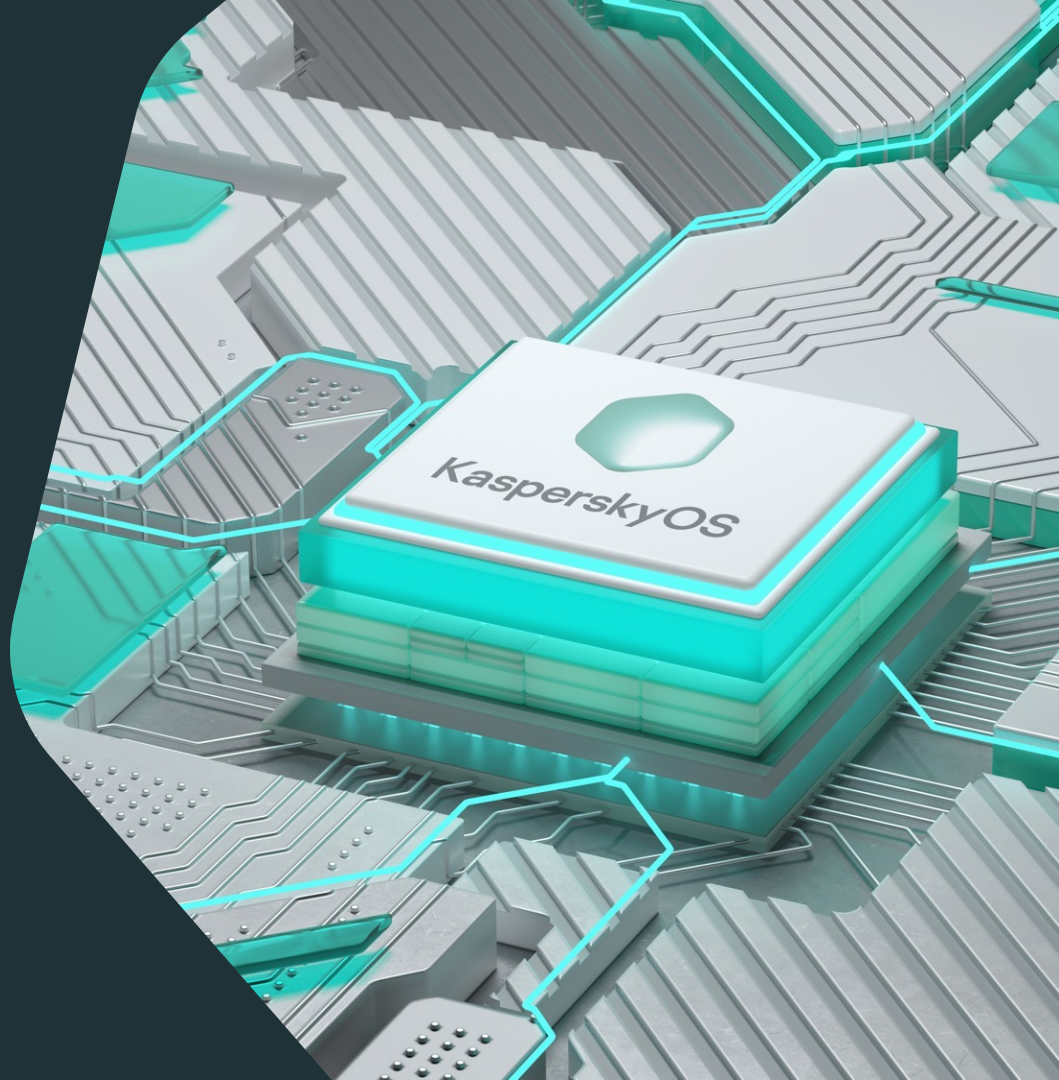
KasperskyOS

Анна
20+ лет в ИТ

Виртуализация,
архитектура,
микроядро

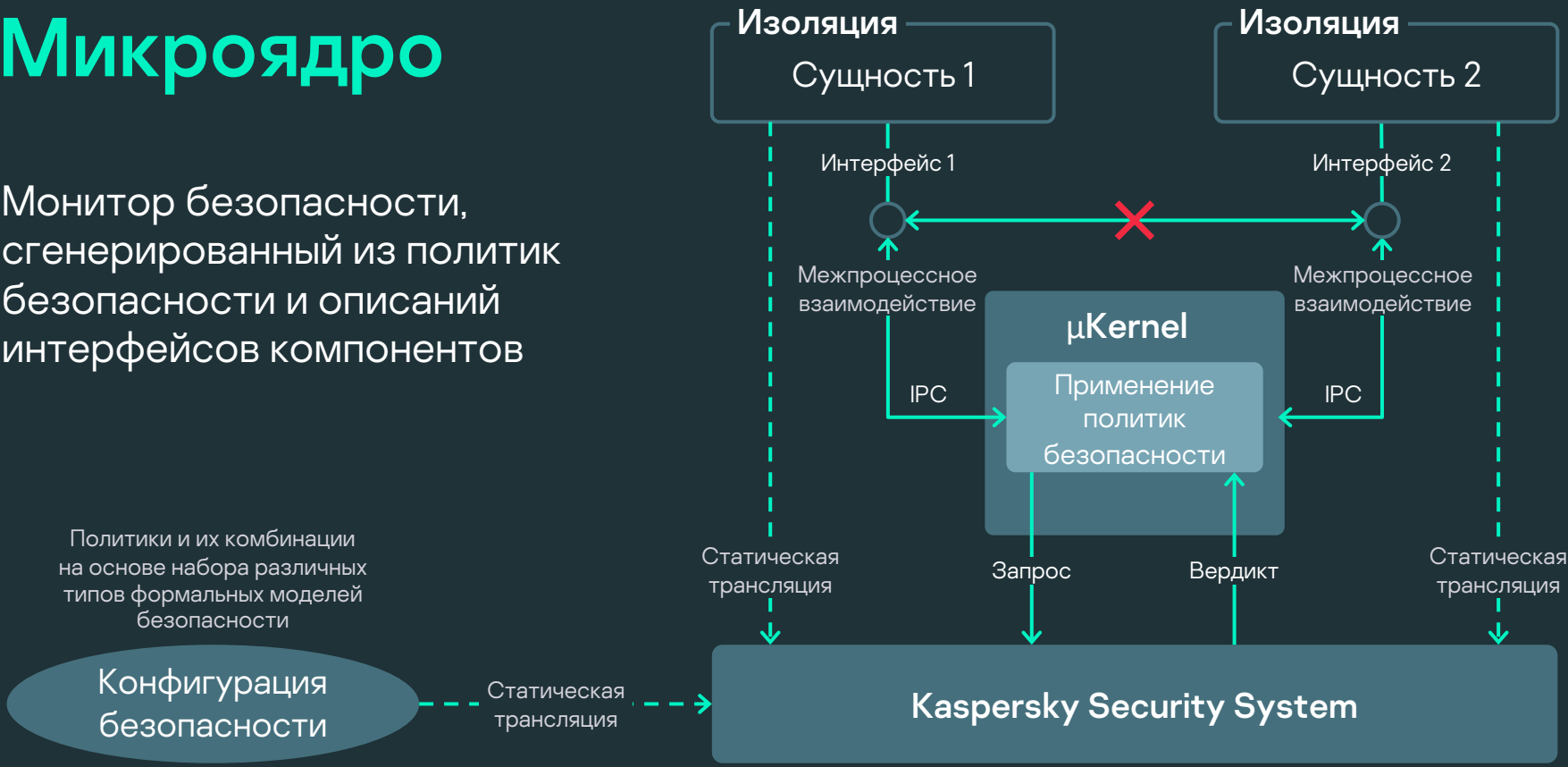
KasperskyOS – операционная система собственной разработки «Лаборатории Касперского»

- Изначально разрабатывается по принципам “secure by design”
- В основе лежит микроядро собственной разработки для снижения объема доверенного кода
- Имеет независимую подсистему вычисления вердиктов безопасности на основе политик – Kaspersky Security Monitor



Микроядро

Монитор безопасности, сгенерированный из политик безопасности и описаний интерфейсов компонентов



Составляющие качества как процесса

Динамический анализ (и хорошее тестовое покрытие)

Фаззинг

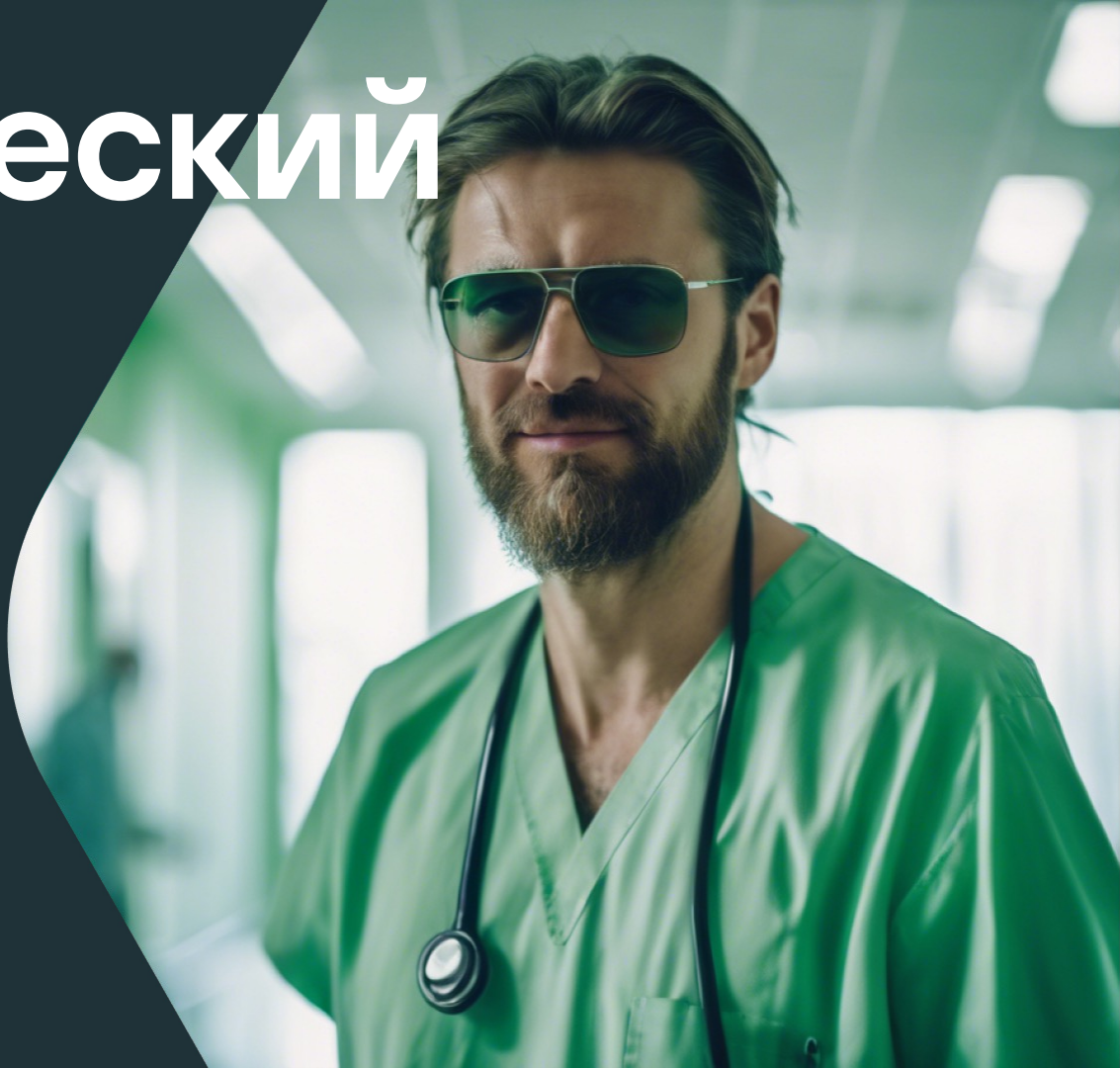
Статический анализ (и guideline по «безопасному» кодингу)

Харденинги

Пен-тест и разбор результатов пен-теста

Динамический анализ

Что могут санитары



Санитары в user-space


```
gcc -fsanitize=address,undefined,float-divide-by-zero,nullability...
```

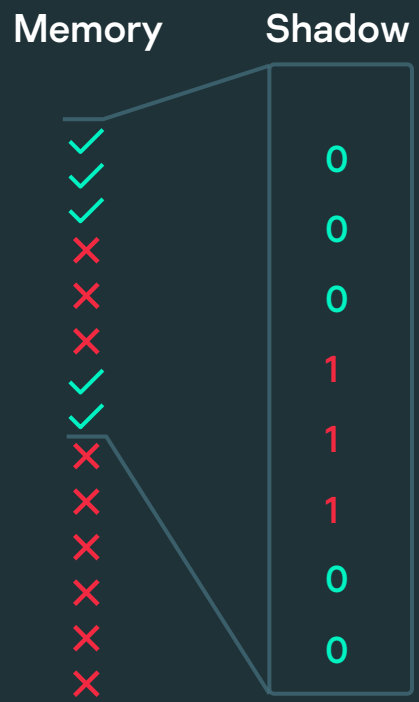
```
[UBSAN] [ERROR] Undefined behaviour at XXX/nsaccess.c:755:9
[UBSAN] [ERROR] misaligned address access at 0x50c2be01301 for the type 'UINT32' (aka 'unsigned int'),
the alignment should be 4
...
tests_kos_queue: [UBSAN] [ERROR] Undefined behaviour at XXX/queue_test.c:266:5
tests_kos_queue: [UBSAN] [ERROR] Operation with pointer 0x2c8500 gives overflow: 0x2c8480
```

ASAN (aka Address Space Sanitizer)


```
char *MemToShadow(rtl_uintptr_t addr)
{
   char *shadow = asanOps.getShadow();
   return &shadow[(addr - asanOps.getMem())
    / ASAN_SHADOW_SCALE_FACTOR];
}
```

Данные

- На стеке
- В куче
- Глобальные данные
- Собственные буферы



Санитары в ядре

Не совсем так,
как привыкли

Куча и свои memcpy — обучение санитаров своему runtime

Нежданчик noreturn

Shadow-регион — проблемы размещения

ASAN + куча = друзья навек


```
void AsanKmallocRedzone(AllocHeader *buf, rtl_size_t origSize)
{
    rtl_uintptr_t addr = (rtl_uintptr_t) buf;
    ShadowSet(addr, sizeof(AllocHeader), SHADOW_KMALLOC_REDZONE);

    addr += sizeof(AllocHeader);
    ShadowSet(addr, origSize, SHADOW_VALID);
}

void AsanKmallocFree(AllocHeader *buf, rtl_size_t size)
{
    /* Check for double free. */
    ShadowCheckShouldBe((rtl_uintptr_t) buf, SHADOW_KMALLOC_REDZONE);
    ShadowSet((rtl_uintptr_t) buf, size, SHADOW_KMALLOC_FREE);
}
```

ASAN prolog — обновляет shadow для локальных данных на стеке


```
0xffffc00001072b00 <+0>:  push   %r15
// сохранение других регистров
0xffffc00001082713 <+35>: movabs $0xdfffe00000000000,%rbx // shadow offset
0xffffc0000108271d <+45>: sub    $0x58,%rsp
0xffffc00001082721 <+49>: lea    0x10(%rsp),%rax
0xffffc00001082735 <+69>: shr    $0x3,%rax // shadow = stack_addr/8 + shadow_offset
0xffffc0000108273e <+78>: mov    %rax,0x8(%rsp)
0xffffc00001082743 <+83>: add    %rbx,%rax
0xffffc0000108274f <+95>: movl   $0xf1f1f1f1,(%rax) // помечает стековые данные в shadow
0xffffc00001082755 <+101>:movl   $0xf3f30000,0x4(%rax) // границы данных
```

А если нет epilog?


```
void ThreadDispatchException(
                                                    Thread *current,
                                                    HalTrapFrame *frame,
                                                    ExceptionInfo *info)
{
        /* Для возврата в ExceptionPrologue изменяем PC */
        frame->pc = userHandler;

        ThreadLeaveTrap(current, frame);
        noreturn;
}
```

Фаззинг

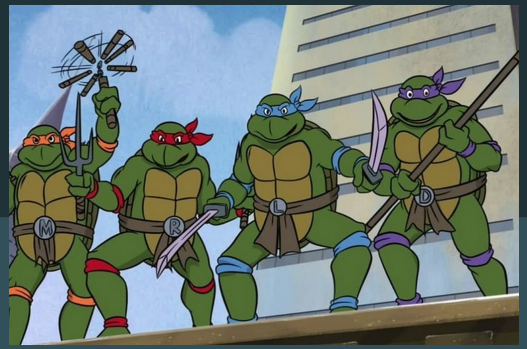
Как не делать ничего и повышать при этом ~~покрытие~~ доверие

Фаззинг

Стоит только запустить,
и баги сразу же найдутся

```
extern "C"
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size)
{

   imgSize = Size;
   img = (rtl_uint8_t *)(rtl_uintptr_t)Data;

   rc = KnElfCreate (img, imgSize,
                     &relocBase, &elfSegs,
                                       &symIndex, &symSize,
                                       &hdrData, &hdrSize);
```

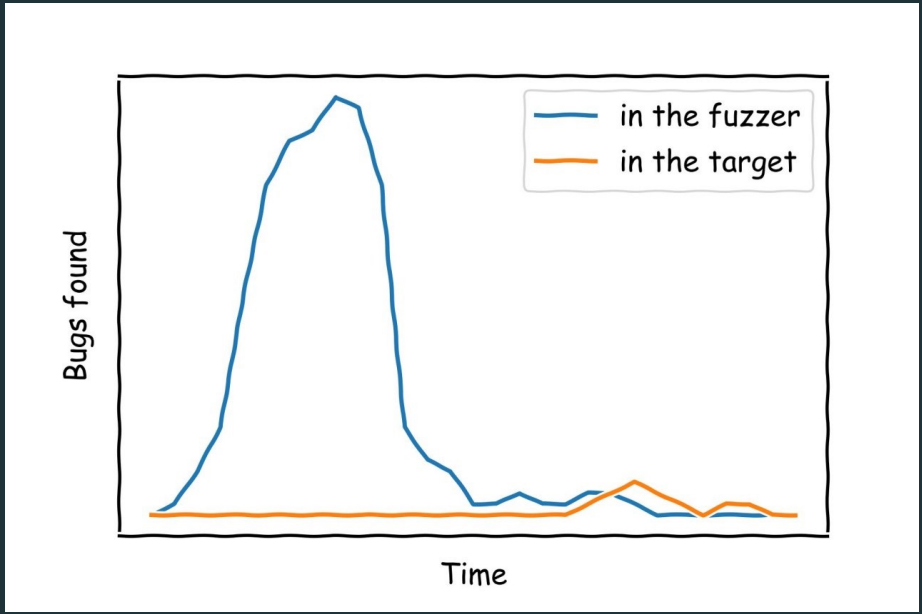
Реальность фаззинга



```
size_t MutateElf(uint8_t *Data, size_t Size)
{
  Elf_Ehdr *ehdr = (Elf_Ehdr *)Data;

  rtl_memcpy(ehdr->e_ident, ELFMAG, SELFMAG);

  ehdr->e_ident[EI_CLASS] = ELFCLASS_EXPECTED;
  ehdr->e_ident[EI_DATA] = ELFDATA_EXPECTED;
  ehdr->e_type = ET_EXEC;

  ehdr->e_ehsize = sizeof(Elf_Ehdr);
  ehdr->e_phentsize = sizeof(Elf_Phdr);
  ehdr->e_shentsize = sizeof(Elf_Shdr);
```

Фаззинг:

«стоит только запустить» ???



Фаззинг в ядре

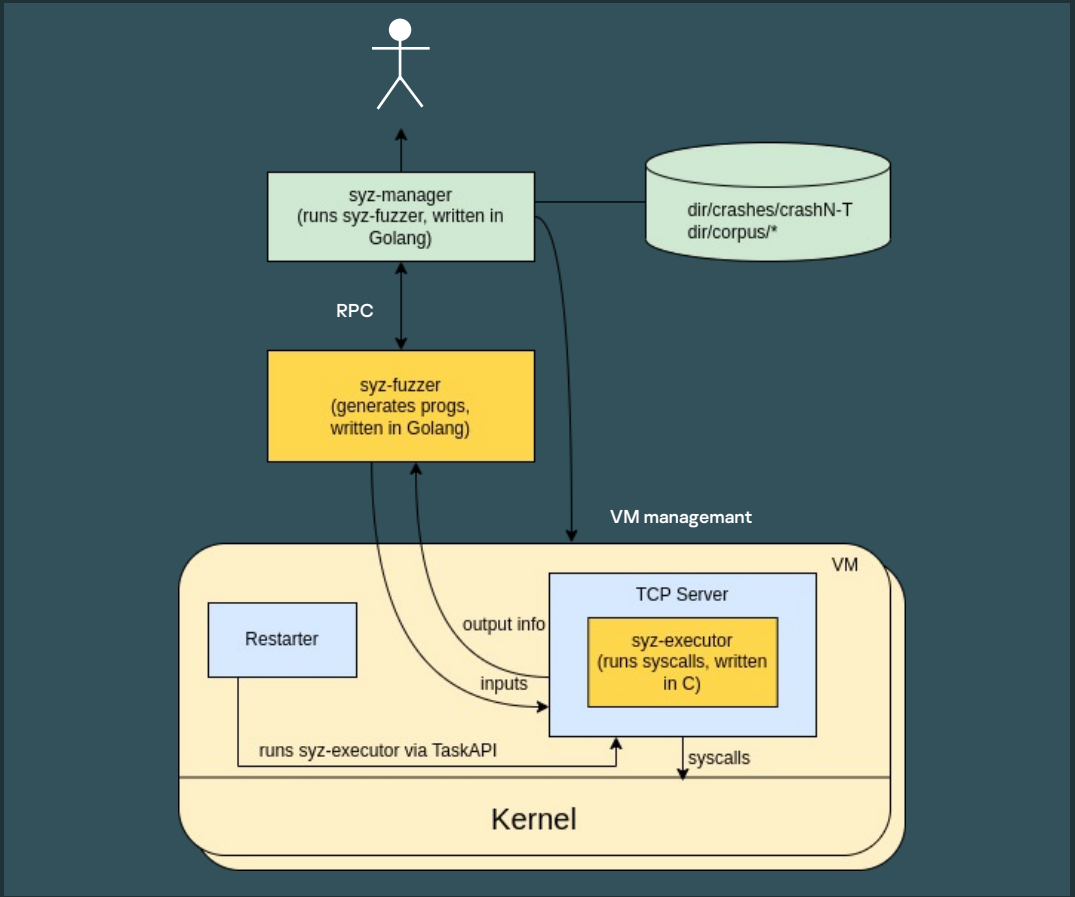
... стоит только запустить!... ☹

Фаззинг не библиотека

Свой coverage, transport, crash detector

Использует описания

Архитектура Syzkaller



Кодогенерация для syzkaller

Интерфейс KasperskyOS

```
MdlClone(in Handle originHandle, in UInt64 offset, in UInt64 length,
         out Handle cloneHandle, out TrRetcode rc);
```

syzkaller syzlang

```
kl_core_VMM_MdlClone_req {
  _base      idl_aligned8[nk_message]
  originHandle   idl_aligned8[nk_handle_desc_t_template[nk_handle_t_VMM]]   (in)
  offset     idl_aligned8[nk_uint64_t]
  length     idl_aligned8[nk_uint64_t]
} [packed]
kl_core_VMM_MdlClone_res {
  _base      idl_aligned8[nk_message]
  cloneHandle idl_aligned8[nk_handle_desc_t_template[nk_handle_t_VMM]]   (out)
  rc     idl_aligned4[nk_sint32_t]
} [packed]
kl_core_VMM_MdlClone_wrapper(req ptr[in, kl_core_VMM_MdlClone_req], res ptr[out,
kl_core_VMM_MdlClone_res])
```

Сгенерированные syzkaller'ом программы

C code

```
// ...
  *(uint16_t*)0x20000100 = 0;
  *(uint16_t*)0x20000102 = 0;
  STORE_BY_BITMASK(uint32_t, , 0x20000104, 0, 0, 28);
  STORE_BY_BITMASK(uint32_t, , 0x20000107, 0, 4, 3);
  STORE_BY_BITMASK(uint32_t, , 0x20000107, 0, 7, 1);
  *(uint32_t*)0x20000108 = 0x349b449c;
  *(uint32_t*)0x2000010c = 0;
  *(uint64_t*)0x20000110 = 0;
  *(uint32_t*)0x20000118 = r[1];
  *(uint32_t*)0x2000011c = 0xdba1e6d2;

  ((intptr_t(*)(intptr_t, intptr_t, intptr_t, intptr_t, intptr_t))CAST(
     kl_core_VMM_MdlClone))(r[0], 0x20000100, 0, 0x20000240, 0);
//...
```

Языковое разнообразие в работе с Syzkaller

1. Go как язык syzkaller
2. C как язык KasperskyOS kernel и syz-executor
3. Haskell как язык кодогенерации, чтобы генерировать syzlang
4. Ассемблер как язык сбора покрытия

4

Ассемблер как язык сбора покрытия

```
var arches = map[string]Arch{
                targets.AMD64: {
                                callLen:     5,
                                opcodes:     [2]byte{0xe8, 0xe8},
                }
}

for i, opcode := range data {
   if opcode != arch.opcodes[0] && opcode != arch.opcodes[1]       {
        continue
   }
   i -= arch.opcodeOffset
   if i < 0 || i+arch.callLen > len(data) {
     continue
```

Фаззинг не прост

1. +1 в недрах подсистемы управления памятью
2. Переиспользование неосвобожденных на fail путях ресурсов
3. Найденные гонки в планировщике

... однако, какие ж вкусные баги

Статический анализ

В чем подвох?

```
Retcode SchedSetClass(Thread *thread, SchedIdx index, rtl_uint32_t priority)
{
    SchedModifyRequest request = {
        .classId = index,
        .priority = priority
    };

    if (index == IdleIndex || index > rtl_countof(schedClasses)
        || priority > schedClasses[index]->prioMax)
          return rcInvalidArgument;

    return SchedModifySched(thread, &request);
}
```

Статические анализаторы в ядре

Нет подвоха!

Нет подвоха!

Нет подвоха, ну почти

```
//V_FUNC_ALIAS, implementation:strdup, function:rtl_strdup
//V_FUNC_ALIAS, implementation:assert, function:rtl_assert
//V_FUNC_ALIAS, implementation:malloc, function:KosMemAlloc
```

PVS-studio нашел, но не svace


```
260: if(uNumBytesToMove && pMe->UB.ptr) { ...
263:    nk_memmove(pDestinationOfMove, pSourceOfMove, uNumBytesToMove);
264: }
265:
266: /* 4. Put the new data in */
267: nk_uint8_t *pInsertionPoint = ((nk_uint8_t *)pMe->UB.ptr) + uInsertionPos;
268: if(pMe->UB.ptr) {
                nk_memmove(pInsertionPoint, NewData.ptr, NewData.len);
```

PVS-studio нашел, но не svace

kosdev-mono/.../UsefulBuf.c:267:1: warning: V1004 [CERT-EXP34-C]
The '((nk_uint8_t *) pMe->UB.ptr)' pointer was used unsafely after it was verified against nullptr. Check lines: 260, 267.

```
260: if(uNumBytesToMove && pMe->UB.ptr) { ...
263:    nk_memmove(pDestinationOfMove, pSourceOfMove, uNumBytesToMove);
264: }
265:
266: /* 4. Put the new data in */
267: nk_uint8_t *pInsertionPoint = ((nk_uint8_t *)pMe->UB.ptr) + uInsertionPos;
268: if(pMe->UB.ptr) {
                nk_memmove(pInsertionPoint, NewData.ptr, NewData.len);
```

Svace нашел, но не PVS-studio


```
#define BITSET_ARB_BIT_INTO_UNIT(set,bit) ({            \
  nk_typeof(set) s = (set);                             \
  nk_size_t r = (bit) / BITSET_ARB_UNIT_BITS(s->data[0]); \
  nk_assert(r <= nk_array_size(s->data));               \
  r; })
```

Svace нашел, но не PVS-studio


```
#define BITSET_ARB_BIT_INTO_UNIT(set,bit) ({            \
  nk_typeof(set) s = (set);                             \
  nk_size_t r = (bit) / BITSET_ARB_UNIT_BITS(s->data[0]); \
  nk_assert(r <= nk_array_size(s->data));               \
  r; })
```

Харденинги

Как защититься наверняка?

Рекомендуемая реализация

1. Опции **-D_FORTIFY_SOURCE =2 - O 2** компилятора при использовании glibc, начиная с 2.3.4. Они позволяют детектировать некоторые случаи переполнения буфера во время выполнения и закрыть приложение
2. Опция **-fstack-protector --param ssp-buffer-size=4** компилятора GCC до версии 4.9 или опция **-fstack-protector-strong** компилятора GCC версии 4.9. и выше и LLVM/Clang 3.7 и выше
3. Опция **-fstack-clash-protection** компилятора GCC версии 8.1 и выше. Опция реализует защиту от атак класса stack clash
4. Опция **-Wl,-z,separate-code** линковщика Binutils версии 2.30 и выше. Опция выделяет невыполняемые области сегмента PT_LOAD в отдельный сегмент с исключением доступа на выполнение

Харденинги в ядре

Полный комплект для DIY

Специальные опции компилятора

Специфичные аппаратные фичи

Сложные тесты и 5+ архитектур

Харденинги в ядре

Полный комплект для DIY

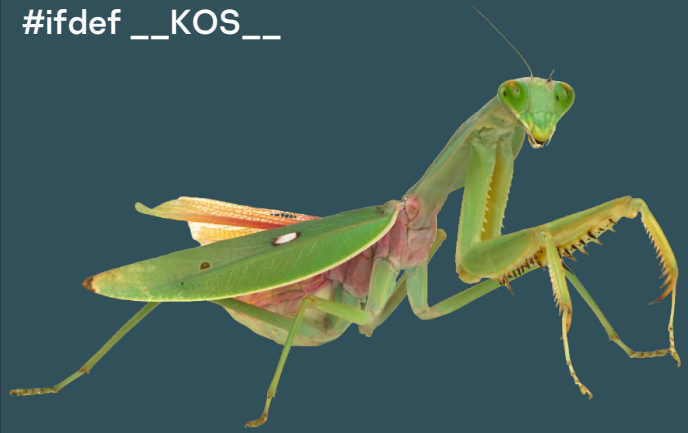
Свобода творчества для повышенных гарантий безопасности!

Специальные опции компилятора

Специфичные аппаратные фичи

Сложные тесты и 5+ архитектур

```
void *__stack_chk_guard = 0;

static void __attribute__ ((constructor))
__guard_setup (void)
{
#ifdef __KOS__
```

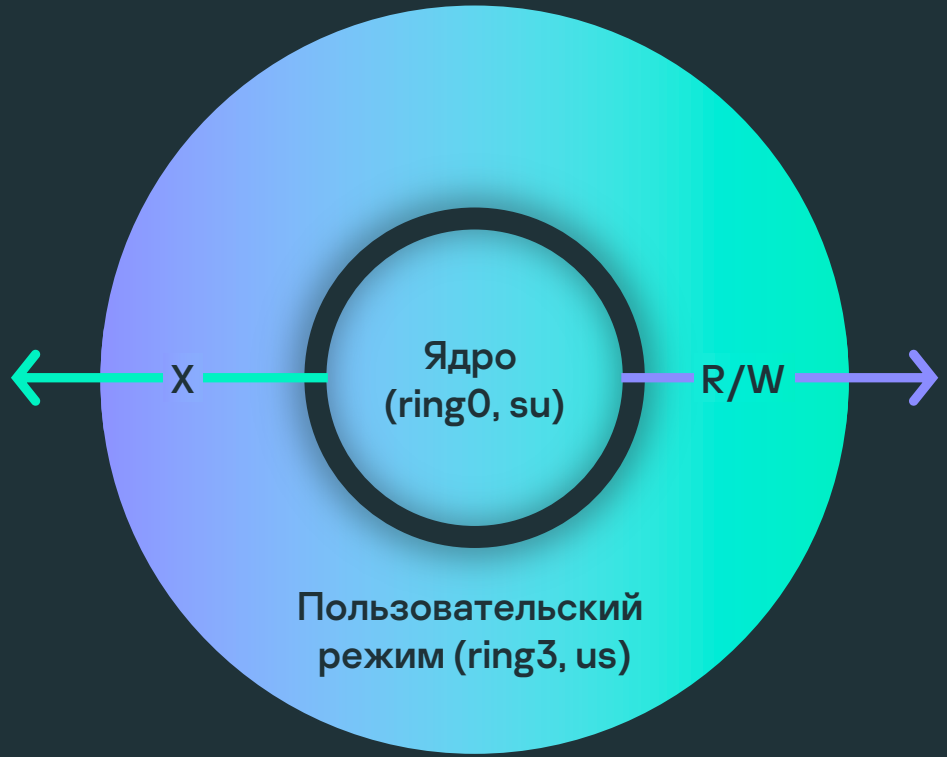


```
#else
```

Изображение: Freepik/kuritafsheen77

Intel SMEP/Arm PXN + Intel SMAP/Arm PAN

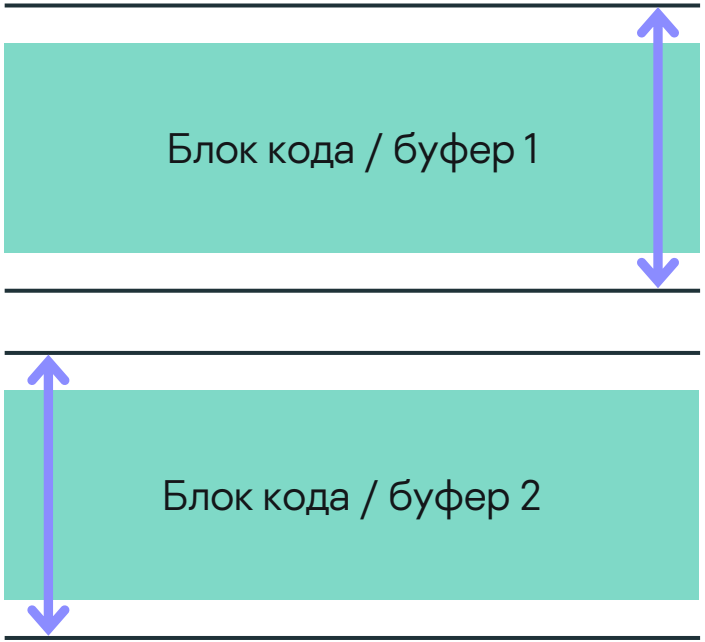
PXN/SMEP: нельзя просто так взять и исполнить код в user-space страницах памяти



PAN/SMAP: нельзя просто так взять и прочитать/записать данные в user-space страницах

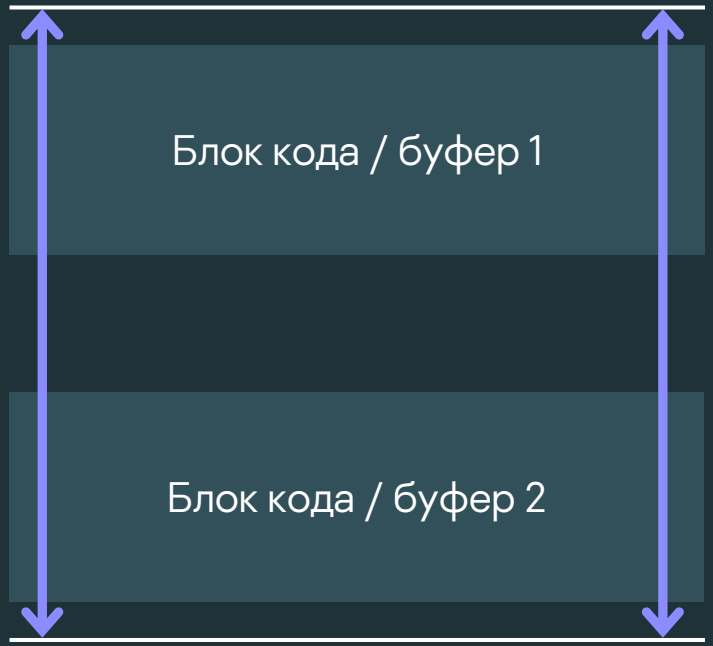
KASLR (Kernel address space randomization)

«Дрожание буферов»



KasperskyOS

Полная рандомизация



Problem statement

Страницы одновременно writable+executable идеальны для payload'а

Давайте не будем разрешать создавать W+X страницы?

приложениям → Но как же JIT?

и ядру с драйверами → Но как же BPF?

KasperskyOS

Давайте не будем разрешать W+X страницы?

приложениям → А отдельным программам, например с JIT, через монитор безопасности разрешим

и ядру с драйверами → А драйверы — в user-space, а ядро — микро. Запрещаем

Пен-тест и его результаты

Как понять, что на правильном пути

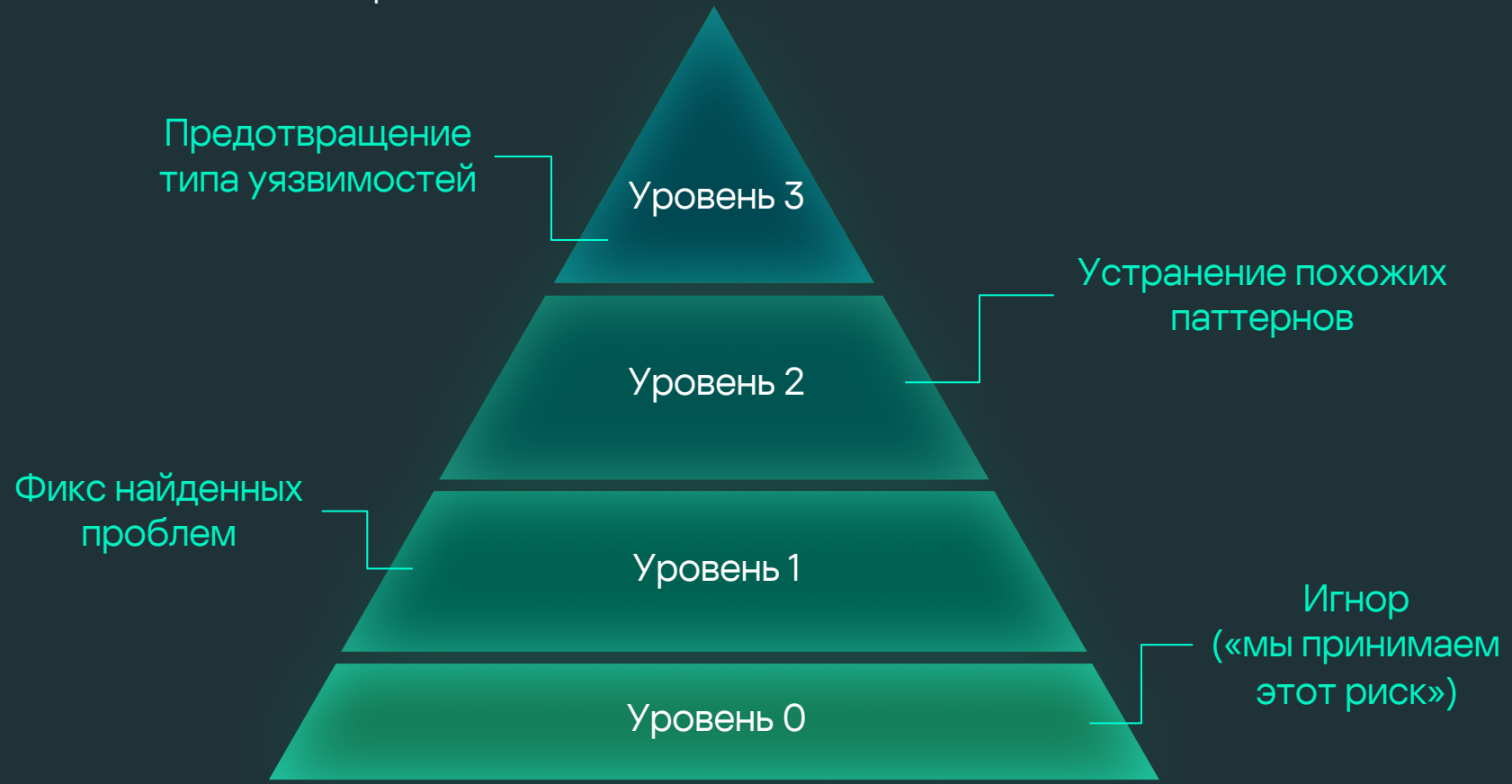
— Не знаете ли вы, как мне выйти отсюда?

— Это зависит от того, куда ты хочешь попасть, — ответил Кот.

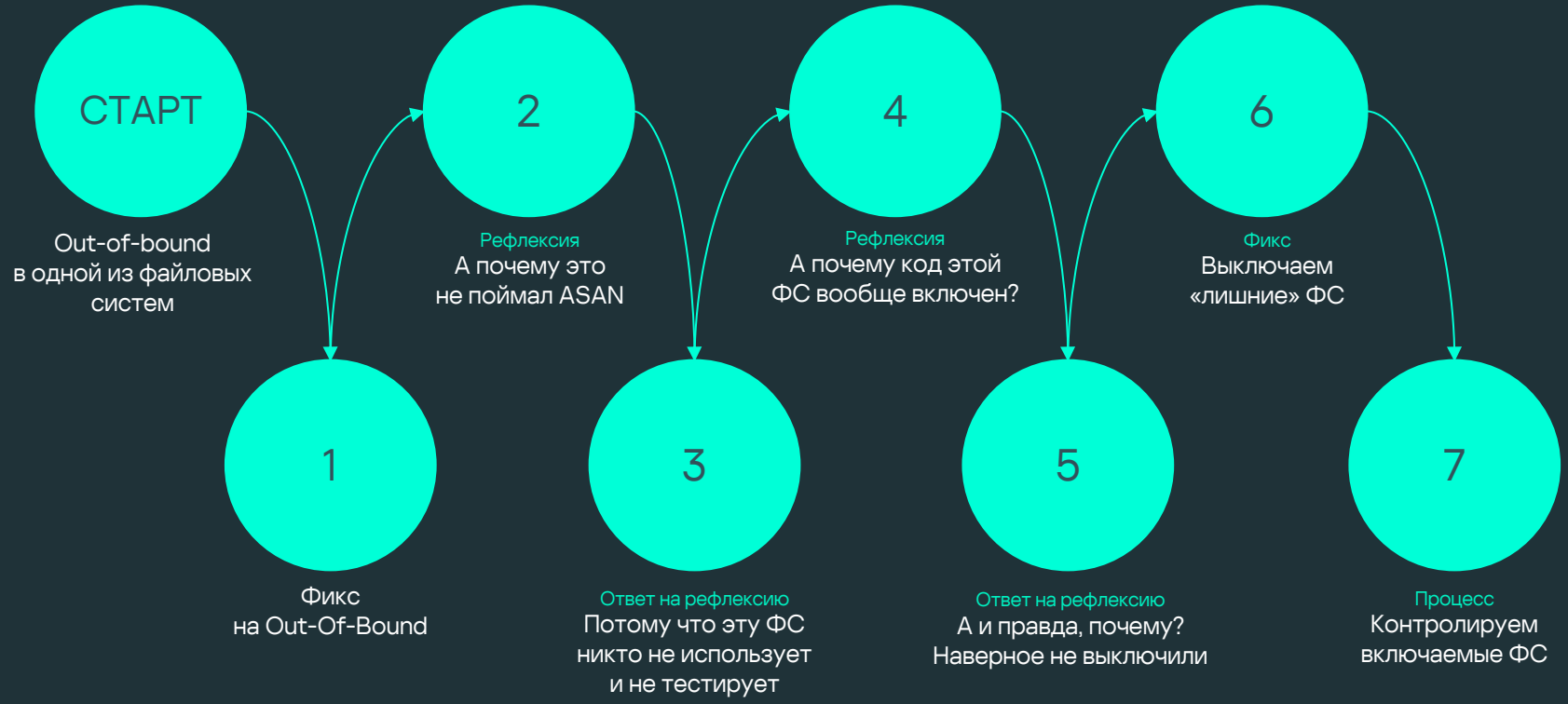
— Мне, в общем-то, все равно... — начала Алиса.

— Значит, тебе все равно, в какую сторону идти, — перебил ее Кот.

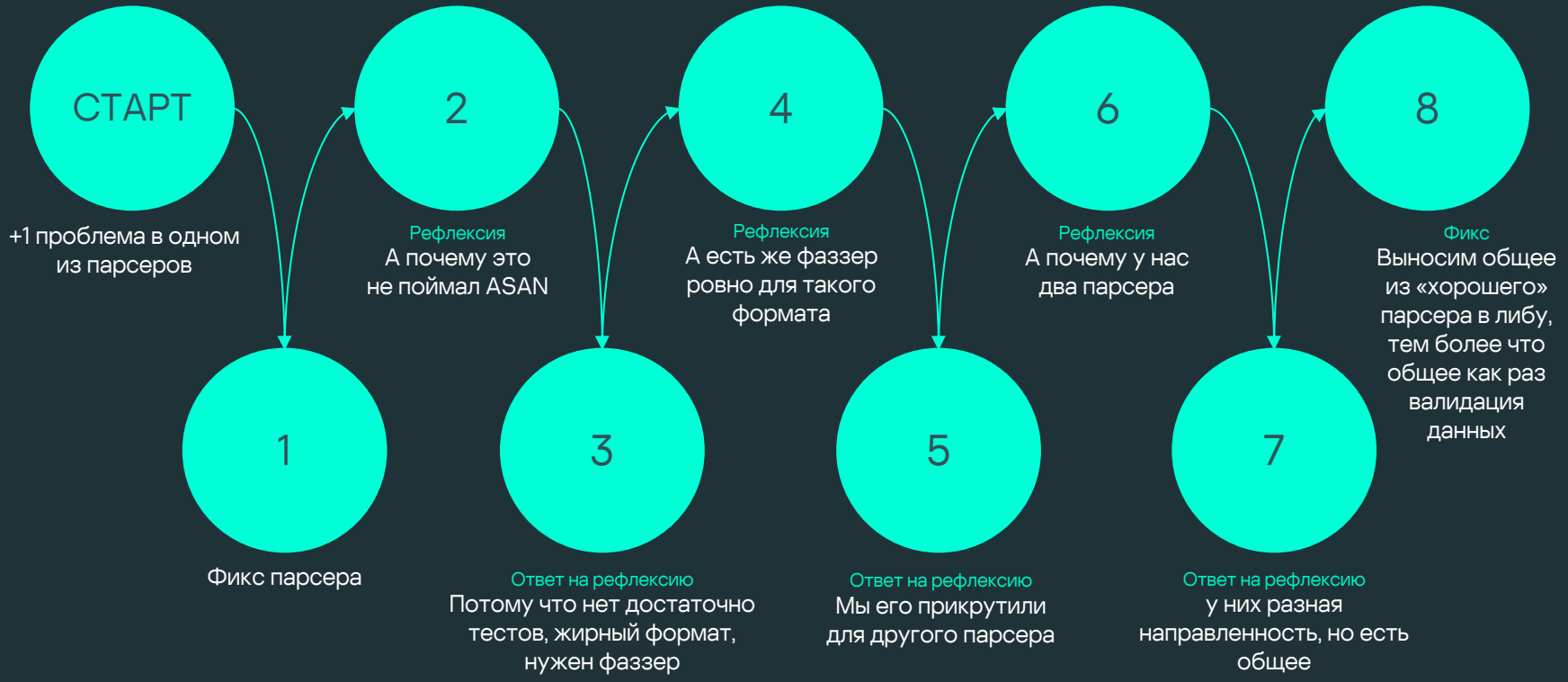
Реагирование на пен-баги: пирамида



Разбор пен-теста (1)



Разбор пен-теста (2)



Выводы

Защита ядра дороже, чем пользовательского режима

Свое ядро не всегда легко стыкуется с инструментами для Linux kernel или пользовательского режима

Однако, в своем ядре может быть больше способов защиты, и они могут быть креативнее

Спасибо!

А мы — молодцы ☺

Анна Мелехова

Руководитель группы разработки защитных решений безопасной платформы

Anna.Melekhova@kaspersky.com

KasperskyOS