پیشرفت های اخیر در فازینگ

شهریار جلایری

معرفی

- شهریار جلایری
- بیش از ۸ سال تجربه تحقیق در زمینه امنیت
  - *مهندسی معکوس، توسعه اکسپلویت، طراحی نرم‌افزار امنیتی*
  - *تمرکز بر روی کشف اتوماتیک آسیب پذیری و فازینگ*

رئوس مطالب

- مقدمه‌ای بر فازینگ
- حوزه‌های تمرکز
- بررسی KFUZZ
- پرسش و پاسخ

مقدمه‌ای بر فازینگ

- فازینگ چیست؟
- تقسیم بندی فازرها (Dumb, Generational, …)
  - *Peach, zzuf, SPIKE, Radamsa*

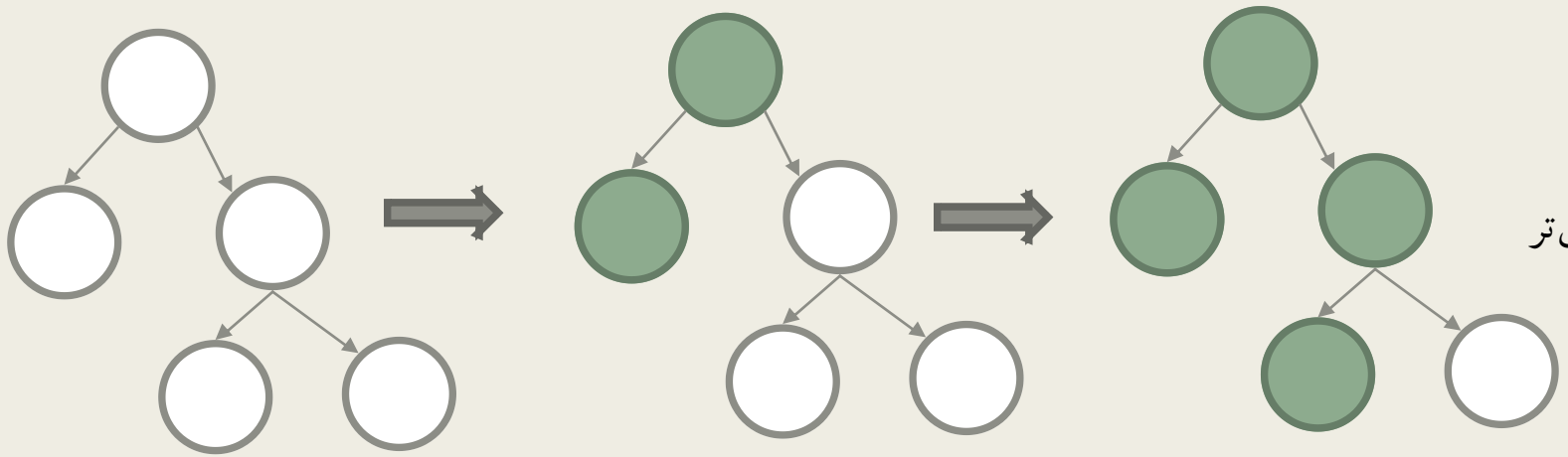
An Empirical Study of the Reliability of UNIX Utilities
Barton P. Miller

حوزه های تمرکز

- افزایش پویش کد (Code Coverage Maximization)
- زمانبندی و تقسیم نیروی محاسباتی (Scheduling)
  - *بهینه سازی انتخاب ورودی*

افزایش پوشش کد (Code Coverage Maximization)

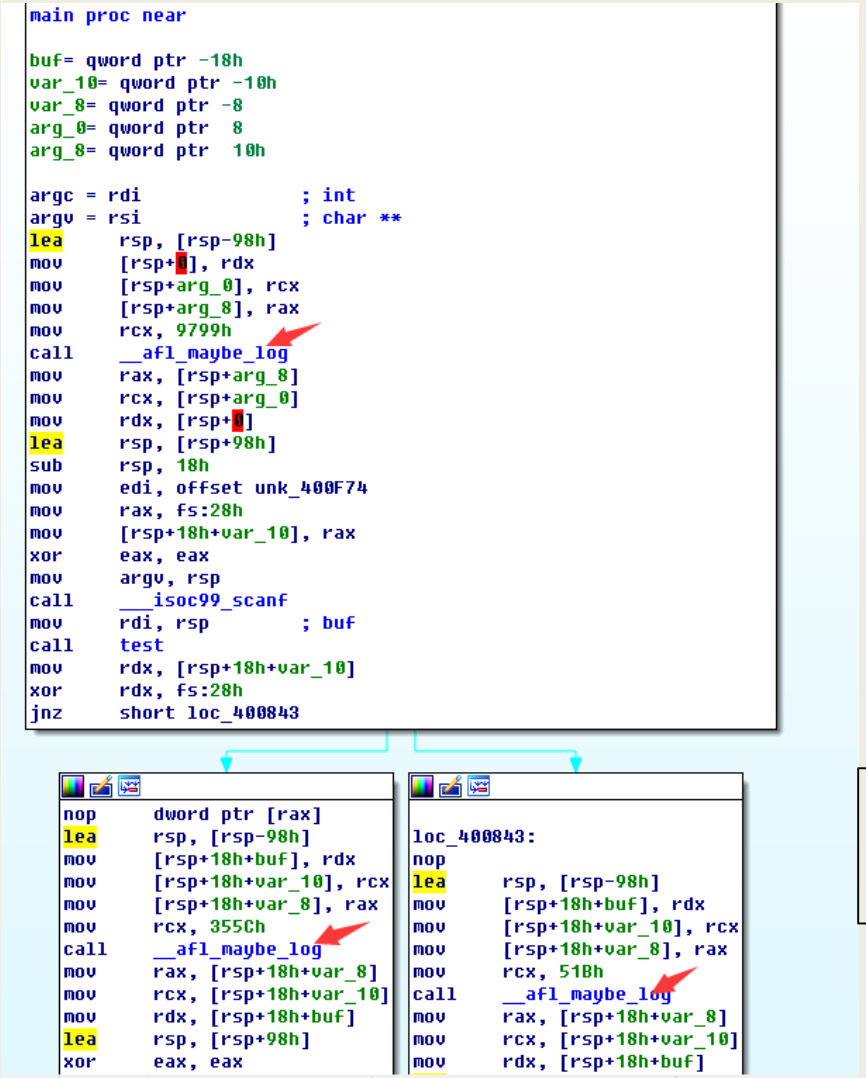
- فازینگ بدون حلقه بازخورد (Feedback Loop)
- ظهور فازرهای مبتنی بر بازخورد (Guided BF)
  - *Bunny the Fuzzer, AFL, LibFuzzer*



نتیجه :

- کارایی بالا
- افزایش پوشش
- کشف آسیب پذیری های عمیق‌تر

افزایش پوشش کد – AFL



- تزیق کد در زمان کامپایل
- ثبت انتقال میان یال ها
- ذخیره بازخورد در Bitmap
- قرار دادن Bitmap در حافظه اشتراکی
- تغییر داده ها (Deterministic, Non-deterministic)

```
cur_location = (block_address >> 4) ^ (block_address << 8);
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

Technical "whitepaper" for afl-fuzz
Michal Zalewski

افزایش پوشش کد – معیارها

- معیارهای اندازه گیری پوشش
  - *رابطه میان یال ها (Basic Block Edges) و ورودی*
  - *رابطه میان عمق پشته (Stack Depth) و ورودی*

```
static void getStackDepth(void) {
  size_t p;
  asm("movq %%rsp,%0" : "=r"(p));
  p = 0x8000000000000000 - p;
  if (p > fuzzer::stackDepthRecord) {
      fuzzer::stackDepthRecord = p;
      if (fuzzer::stackDepthBase == 0) {
          fuzzer::stackDepthBase = p;
      }
  }
}
```

libFuzzer-gv: new techniques for dramatically faster fuzzing
Guido Vranken

افزایش پوشش کد – چالش‌ها

```
1  int main(void) {
2    config_t *config = read_config();
3    if (config == NULL) {
4      puts("Configuration syntax error");
5      return 1;
6    }
7    if (config->magic != MAGIC_NUMBER) {
8      puts("Bad magic number");
9      return 2;
10   }
11   initialize(config);
12
13   char *directive = config->directives[0];
14   if (!strcmp(directive, "crashstring")) {
15     program_bug();
16   }
17   else if (!strcmp(directive, "set_option")) {
18     set_option(config->directives[1]);
19   }
20   else {
21     default();
22   }
23
24   return 0;
25 }
```

- چالش‌های روبروی فازرهای مبتنی بر بازخورد
  - *ثابت های عددی (Numeric Constants)*
  - *ثابت*های رشته ای (Strings Constants)
  - محاسبه و مقایسه سی‌آرسی
  - تبدیل رشته به عدد (توابعی چون strtol)

Driller: Augmenting Fuzzing Through Selective Symbolic Execution
Nick Stephens, et al.

افزایش پوشش کد – چالش‌ها (ادامه)

- راه حل های بدون نیاز به محاسبات سنگین (Heavy Weight Analysis)
  - *شکافتن ثابت ها (Constant Splitting)*

```
if (input == 0xabad1dea) {
  /* terribly buggy code */
} else {
  /* secure code */
}
```

```
if (input >> 24 == 0xab){
  if ((input & 0xff0000) >> 16 == 0xad) {
    if ((input & 0xff00) >> 8 == 0x1d) {
      if ((input & 0xff) == 0xea) {
        /* terrible code */
        goto end;
      }
    }
  }
}

/* good code */
end:
```

افزایش پوشش کد – چالش‌ها (ادامه)

```
if(!strcmp(directive, "crash"))
{
  programbug()
}
```

```
if(directive[0] == 'c') {
  if(directive[1] == 'r') {
    if(directive[2] == 'a') {
      if(directive[3] == 's') {
        if(directive[4] == 'h') {
          if(directive[5] == 0) {
            programbug()
}
```

Fuzzing with AFL is an Art
Brendan Dolan-Gavitt

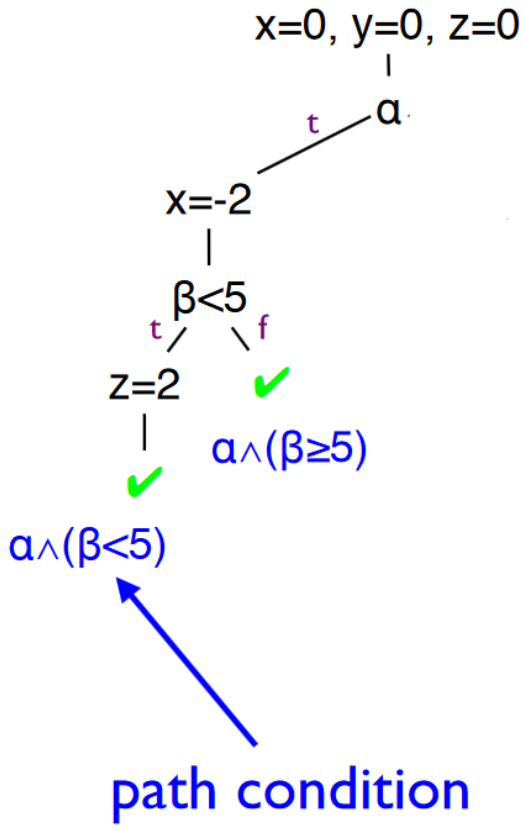
Make AFL-fuzzing wide constants more viable with another llvm pass
AFL Mailinglist

Circumventing Fuzzing Roadblocks with Compiler Transformations
lafintel

افزایش پوشش کد – چالش‌ها (ادامه)

- راه حل های بر پایه اجرای سمبولیک (Symbolic Execution)
  - *اجرای سمبولیک چیست؟*
  - *چالش‌های اجرای سمبولیک*
    - عدم گسترش پذیری
    - انفجار مسیرها (PP, PM)
  - *اجرای سمبولیک گزینشی (Selective Sym Exec)*

```
1. int a = α, b = β, c = γ;
2.                  // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c)  { y = 1; }
9.   z = 2;
10.}
11.assert(x+y+z!=3)
```



SELECT - a formal system for testing and debugging programs by symbolic execution
R. Boyer, B. Elspas, K. Levitt

Symbolic execution and program testing
J. King

Klee: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs
C. Cadar, D. Dunbar, D. Engler

افزایش پوشش کد – چالش‌ها (ادامه)

- مقالات مرتبط

DART: directed automated random testing,
P. Godefroid, N. Klarlund

EXE: Automatically Generating Inputs of Death
C. Cadar, et al.

Driller: Augmenting Fuzzing Through Selective Symbolic Execution
Nick Stephens, et al.

Unleashing MAYHEM on Binary Code
Sang Kil Cha, et al.

S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems
Vitaly Chipounov, Volodymyr Kuznetsov, George Candea

افزایش پوشش کد – تولید ساختار ورودی

- روش های تولید ورودی پیچیده به شکل اتوماتیک
  - *مهندسی معکوس ساختار ورودی*
  - *استفاده از یادگیری ماشین*
  - *استفاده از نیروی انسانی*

Automatic reverse engineering of data structures from binary execution
Z Lin, et al.

dynStruct: An automatic reverse engineering tool for structure recovery and memory use analysis
Daniel Mercier, Aziem Chawdhary, Richard Jones

Active Learning of Input Grammars
Matthias H¨oschele, et al.

Learn&Fuzz: Machine Learning for Input Fuzzing
Patrice Godefroid, Hila Peleg, Rishabh Singh

Not all bytes are equal: Neural byte sieve for fuzzing
Mohit Rajpal, William Blum, Rishabh Singh

Rise of the HaCRS: Augmenting Autonomous Cyber Reasoning Systems with Human Assistance
Yan Shoshitaishvili, et al.

زمانبندی (Scheduling)

- نیاز به زمانبندی و جهت دهی
- روش های زمانبندی
  - *مبتنی احتمال*
  - *مبتنی بر پوشش کد*
  - *مبتنی بر معیارهای پیچیدگی کد*
- حوزه‌های زمانبندی
  - *زمانبندی برپایه test-case*
  - *زمانبندی بر پایه mutatorها*

```
720     switch (*pdwInitialStage) {
721
722         case STAGE_BF1  : *pdwInitialStage = 0; goto bitflip1_stage;
723         case STAGE_BF2  : *pdwInitialStage = 0; goto bitflip2_stage;
724         case STAGE_BF4  : *pdwInitialStage = 0; goto bitflip4_stage;
725         case STAGE_BF8  : *pdwInitialStage = 0; goto bitflip8_stage;
726         case STAGE_BF32 : *pdwInitialStage = 0; goto bitflip32_stage;
727         case STAGE_AR8  : *pdwInitialStage = 0; goto arith8_stage;
728         case STAGE_AR16 : *pdwInitialStage = 0; goto arith16_stage;
729         case STAGE_AR32 : *pdwInitialStage = 0; goto arith32_stage;
730         case STAGE_INT8 : *pdwInitialStage = 0; goto intrest8_stage;
731         case STAGE_INT16: *pdwInitialStage = 0; goto intrest16_stage;
732         case STAGE_INT32: *pdwInitialStage = 0; goto intrest32_stage;
733         case STAGE_HVC  : *pdwInitialStage = 0; goto havoc_stage;
734         case STAGE_SPC  : *pdwInitialStage = 0; goto splice_stage;
735         case STAGE_RMQ  : *pdwInitialStage = 0; goto rand_mix_q_stage;
736         case STAGE_RMF  : *pdwInitialStage = 0; goto rand_mix_f_stage;
737         case STAGE_IM   : *pdwInitialStage = 0; goto input_merg_stage;
738         case STAGE_DIC  : *pdwInitialStage = 0; goto dict_stage;
739         case STAGE_ISB  : *pdwInitialStage = 0; goto input_size_bf_stage;
740     }
```

زمانبندی (ادامه)

- زمانبندی مبتنی بر احتمال
  - *زمانبندی بر اساس تعداد Crashها*
  - *زمانبندی بر اساس مقدار پوشش*

$$d_i = \frac{u_i}{t_i}$$

d = Crash Density
u = Unique Crashes
t = Trials

$$D = \sum_{i=1}^{n} d_i$$

D = All the crash densities

$$p_i = \frac{d_i}{D}$$

Probability-Based Parameter Selection for Black-Box Fuzz Testing
Allen D. Householder, Jonathan M. Foote

Coverage-based Greybox Fuzzing as Markov Chain
Marcel Böhme, et al.

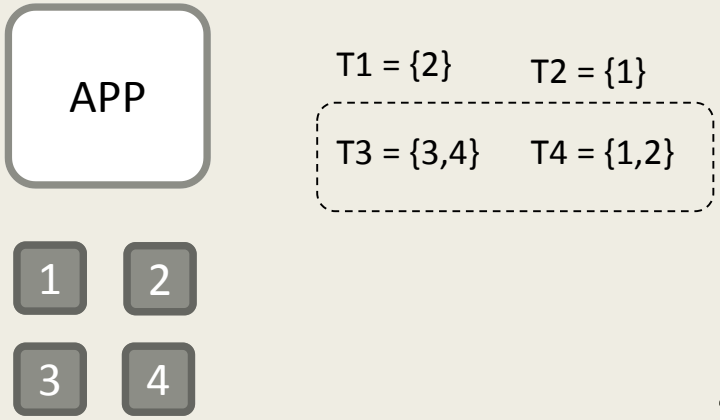
زمانبندی (ادامه)

- زمانبندی مبتنی بر پوشش کد
- زمانبندی مبتنی معیارهای پیچیدگی کد
  - *شاخص کمی برای تعداد مسیرهای خطی مستقل (Cyclomatic Complexity)*
  - *Fan-In Fan-Out*
  - *پیچیدگی دستورالعمل‌ها*

Mining Metrics to Predict Component Failures
Nachiappan Nagappan, et al.

USING COMPLEXITY, COUPLING, AND COHESION METRICS AS EARLY INDICATORS OF VULNERABILITIES
Istehad Chowdhury

زمانبندی – انتخاب ورودی اولیه



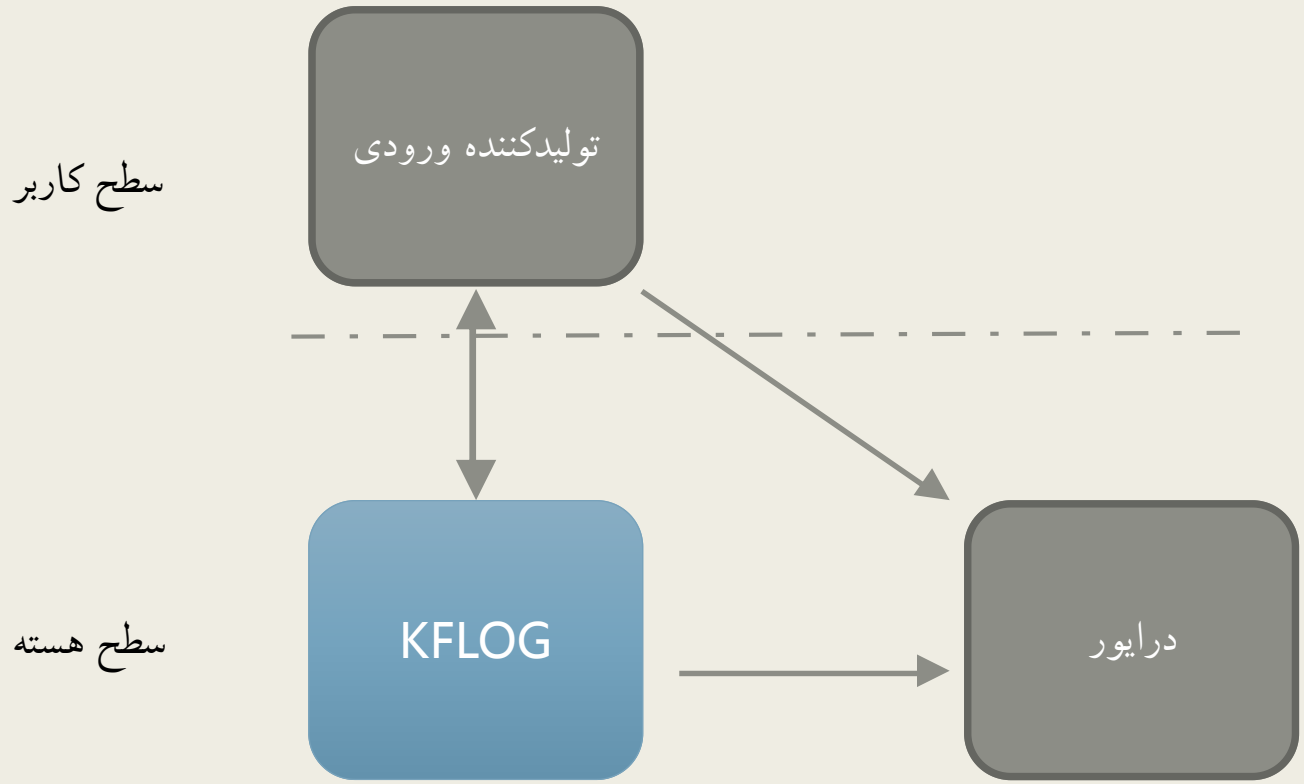
- انتخاب ورودی اولیه (Seed Selection)
  - اندازه گیری پوشش هر قطعه ورودی
  - مسئله پوشش مجموعه (Minimal Set Cover Problem)
    - مسئله NP، قابل حل با استفاده از الگوریتم تقریب حریصانه
- تقلیل فضای قابل فاز کردن هر ورودی با بهره‌گیری از روش‌های ثبت جریان داده
  - Taint Analysis

Optimizing Seed Selection for Fuzzing
Alexandre Rebert, et al.

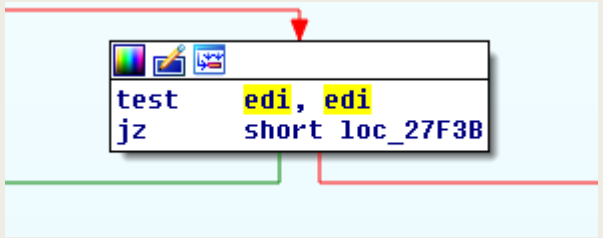
A Taint Based Approach for Smart Fuzzing
Sofia Bekrar, et al.

معرفی KFUZZ

- معماری KFUZZ
  - *فازر هسته ویندوز (مشابه AFL)*
  - *ماژولار*



چالش ها



- ایجاد حلقه بازخورد برپایه انتقال یال ها (Edge Transition)
  - *بازنویسی باینری (Binary Rewriting)*
    - خطاپذیر بودن
    - بازیابی CFG غیر ممکن است!
  - *هوک کردن BBLها*
    - تمییز دادن ارتباطات درایور
    - بلاک‌های کوچک تر ۵ بایت
    - استفاده از Illegal Instruction و هوک کردن IDT
    - استفاده از IDA Pro جهت استخراج بلاک‌ها

```
103         mov     ebx, dword ptr [_g_UsermodeFuzzerPid]
104         cmp     ebx, 0xBAADF00D /* global PID  */
105         jz      __trace_bbl
106         call    PsGetCurrentThreadProcessId
107         cmp     ebx, eax
108         jnz     __no_trace
109
110 __trace_bbl:
111         /* cur_location = (block_address >> 4) ^ (block_address << 8);
112            shared_mem[cur_location ^ prev_location]++;
113            prev_location = cur_location >> 1;
114         */
115         mov     ecx, dword ptr [ebp + 0x0c]     /* BlockId                                  */
116         mov     edx, _g_PrevLocation            /* load up the previous location value      */
117 #if defined(__COMPUTE_FNV_HASH)
118         push    edx
119         push    ecx
120         call    CalculateEdgeHash
121         mov     esi, eax
122 #else
123         xor     edx, ecx                        /* calculate the bitmap offset              */
124         mov     esi, edx                        /* save bitmap offset for later user        */
125 #endif
126
127         mov     eax, esi                        /* move bitmap offset to dividend           */
128         xor     edx, edx                        /* zero out the reminder                    */
129         mov     ebx, 8                          /* set ebx to divisor value                 */
130         div     ebx                             /* perform the division                     */
131         mov     edi, edx                        /* saved modulo result in edi               */
132
133         mov     ebx, _g_EdgeTransitionMap       /* load up the bitmap array                 */
134         mov     dx, word ptr [ebx+eax]          /* load up the target bits                  */
135         bts     dx, edi                         /* set the target bit and save the old value in CF  */
136         mov     word ptr [ebx+eax], dx          /* set back the bitmap target bit to the array      */
137
```

کارایی

- بررسی سربار و سرعت KFLOG

- Native Speed

```
00000003  0.97552210          [KTEST] Counting Primes for 5 seconds...
00000004  5.31115818          [KTEST] Found 16956 prime numbers between 1 and 187394 in 5 seconds
```

- KFLOG Branch Trace enabled (2.44x slowdown)

```
00000418  87.48816681         [KTEST] Counting Primes for 5 seconds...
00000419  92.31293488         [KTEST] Found 6923 prime numbers between 1 and 69830 in 5 seconds
```

کارایی (ادامه)

- تست پوشش دهی
- پیکره های تست کارایی
  - *LAVA* –
  - *EvilCoder* –
  - *CGC* –

```
def generate_pogram(expr_count, input_size, prog_num) :
    generator = GramGenerator()
    generator.add_prod("start",         "expression")
    generator.add_prod("expression",    "buffer condition constant | complex | factor")
    generator.add_prod("complex",       "expression operation expression")
    generator.add_prod("factor",        "( expression ) | expression")
    generator.add_prod("condition",     "== | != | > | <")
    generator.add_prod("operation",     "and | or")
    generator.add_prod("constant",      "CONST")


    expr_inst = list()
    cond_stmt = "if ( %s ) { if (rgp_s[%d] != TRUE) { DbgPrint(\"[RGF-" + str(prog_num) + "] pa
    func_head = "BOOLEAN rgp_s[%d] = { %s };\n%s\n%s"
    func_body = "NTSTATUS RandProg_%d(__in PUCHAR Buffer, __in ULONG BufferLen) { if (BufferLen
    final_block = ""
    stage_init = ""

    for i in range(0, expr_count):
        raw_expr = generator.gen_rand_instance('start')
        raw_expr = raw_expr.replace("and", "&&")
        raw_expr = raw_expr.replace("or", "||")
        raw_expr = replace_constants(raw_expr)
        raw_expr = replace_buffers(raw_expr, input_size)
```

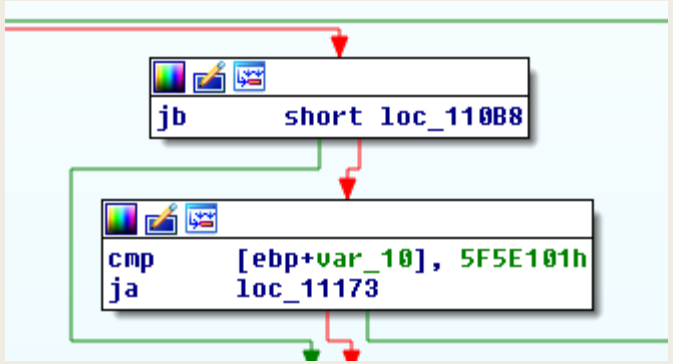
LAVA: Large-Scale Automated Vulnerability Addition
Brendan Dolan-Gavitt, et al.

EvilCoder: automated bug insertion
Jannik Pewny,Thorsten Holz

بهبود سرعت

- عدم نیاز به اسکن کردن bitmap با استفاده از دستور BTC
- بهره گیری از حافظه اشتراکی جهت دسترسی به متغیر EdgeCounter
  - *۱۰۰۰۰ بار دسترسی از طریق حافظه اشتراکی 53709 TICS/174 us*
  - *۱۰۰۰۰ بار دسترسی از طریق I/O درایور 16778493 TICS/50863 us*
- فایل سیستم مسطح بر روی NTFS

چالش ثابت ها

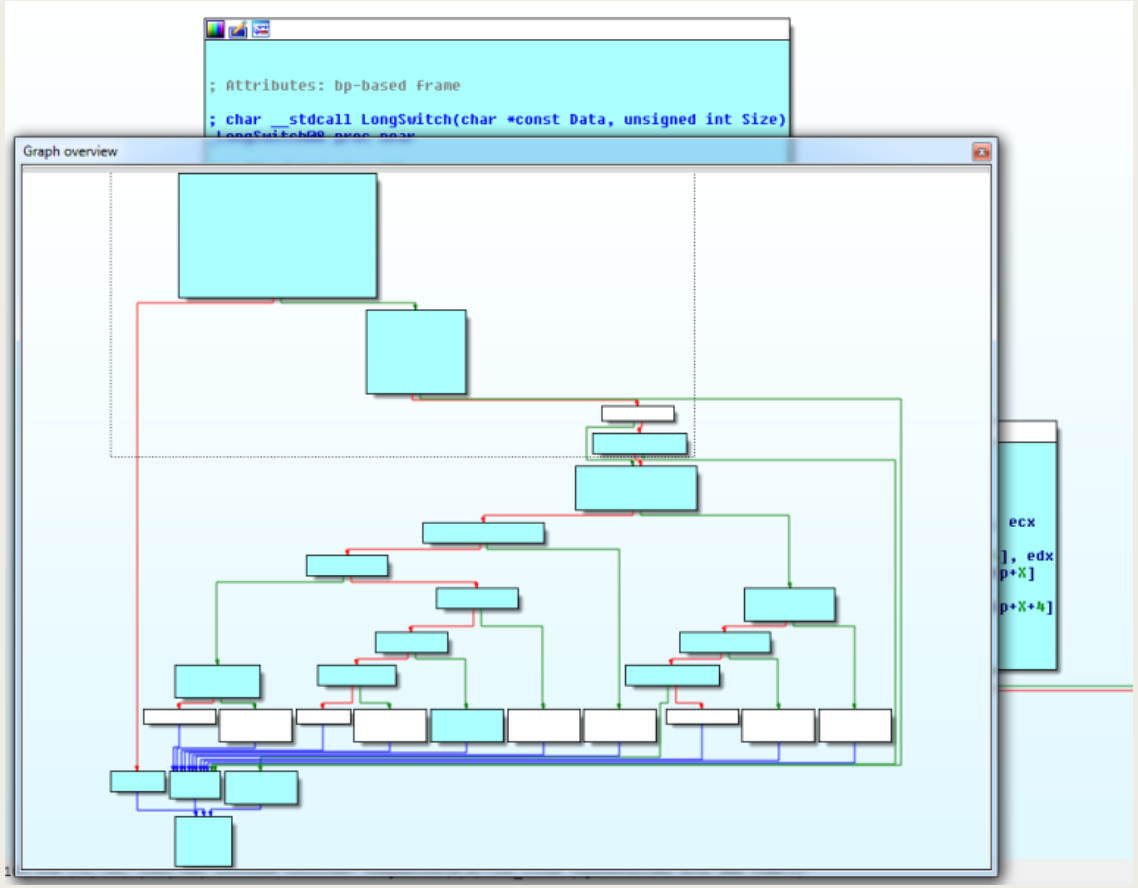


- هوک کردن توابع کار با رشته (strcmp, memcp) جهت دریافت بازخورد
- بهره گیری از آنالیز ایستا
  1. استخراج رشته ها
  2. استخراج ثابت های عددی و شروط مرتبط (CMP با نهاد ثابت)
  3. محاسبه نقیض شروط ثابت های عددی
  4. تزریق ثابت‌ها/نقیض/رشته‌ها در هنگام فاز کردن

```
000010A0  04 00 00 00 00 00 00 00  04 00 00 00 FF FF FF FF                ÿÿÿÿ
000010B0  04 00 00 00 01 00 00 00  04 00 00 00 FF FF FF FF                ÿÿÿÿ
000010C0  07 00 00 00 43 43 43 43  43 43 00 05 00 00 00 42      CCCCCC      B
000010D0  42 42 42 00 0C 00 00 00  63 72 61 73 68 73 74 72  BBB     crashstr
000010E0  69 6E 67 00 1A 00 00 00  5B 4B 54 45 53 54 5D 20  ing     [KTEST]
000010F0  42 61 64 20 6D 61 67 69  63 20 6E 75 6D 62 65 72  Bad magic number
00001100  0A 00 1B 00 00 00 20 20  20 20 5B 25 64 5D 20 20            [%d]
00001110  25 73 20 28 25 73 29 20  61 74 20 30 78 25 78 0A  %s (%s) at 0x%x
```

چالش ثابت ها – نمونه

```
static volatile INT sink;

BOOLEAN
LongSwitch(
    CONST PUCHAR Data,
    SIZE_T       Size
    )
{
    ULONGLONG X;

    if (Size < sizeof(X))
        return FALSE;

    memcpy(&X, Data, sizeof(X));
    switch (X) {

        case 1: sink = __LINE__; break;
        case 101: sink = __LINE__; break;
        case 1001: sink = __LINE__; break;
        case 10001: sink = __LINE__; break;
        case 100001: sink = __LINE__; break;
        case 1000001: sink = __LINE__; break;
        case 10000001: sink = __LINE__; break;
        case 100000001: return TRUE;
    }

    return FALSE;
```



دمو

چالش‌های پیشروی KFUZZ

- تشخیص و دریافت بازخورد از حلقه‌ها
- دریافت بازخورد از دستور CMP
- دخیل کردن آدرس Callee در محاسبه آدرس انتقال در Bitmap
- بهره‌گیری از روش هایی چون اجرای کانکالیک (Concolic Execution)
- ایجاد ساختار ورودی به صورت اتوماتیک (به طور مثال Live Analysis بر روبی ساختار ورودی درایورها)
- استفاده از Taint Analysis جهت فاز کردن بخش‌های مورد استفاده ورودی
- بهره‌گیری از روش‌هایی چون ART جهت تولید ورودی و یا زمانبندی
- بهره‌گیری از تکلنوژی‌هایی چون Intel PT

پایان

- سخن آخر
- پرسش و پاسخ