

人工智能在漏洞发现中的运用之虚与实

Kang Li

kangli.ctf@gmail.com



自我介绍

乔治亚大学教授

Disekt、*SecDawgs* CTF 团队创始人

xCTF 和蓝莲花团队启蒙导师

2016 年 DARPA Cyber Grand Challenge 决赛获奖者



人工智能应用新浪潮

- 由广泛学习和深度学习驱动
- 由海量数据集提供支持



公众什么时候将技术视为人工智能？



在它击败人的时候！

AI & CGC

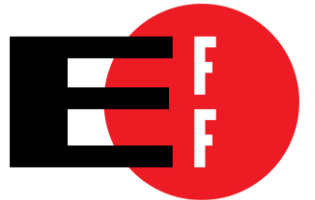


比赛之前



Elon Musk Says DARPA A.I. Hacking Challenge Will Lead to Skynet

He may or may not be joking.



AUGUST 4, 2016 | BY NATE CARDOZO AND PETER ECKERSLEY AND JEREMY GILLULA



Does DARPA's Cyber Grand Challenge Need A Safety Protocol?

比赛之后

DARPA's king Robo-hacker "Mayhem" proves no match for humans

SECURITY AND PRIVACY

🕒 12th August 2016 👤 Matthew Griffin 💬 0

Scoreboard

place	score	team
1	15	PPP
2	14	b1o0p
3	13	DEFKOR
4	12	HITCON
5	11	KaisHack GoN
6	10	LC & BC
7	9	Eat Sleep Pwn Repeat
8	8	binja
9	7	pasten
10	6	9447
11	5	!SpamAndHex
12	4	Shellphish
13	3	Dragon Sector
14	2	侍
15	1	Mayhem

人工智能在安全领域的应用

机器学习已经广泛用于

恶意软件检测

垃圾邮件和网络钓鱼分类

帐户异常检测和日志分析等

机器学习应用的最新变化

数据规模和算法复杂度发生变化

两种“随机”机器学习应用实例



Amico

Esorics 2013

APK 恶意软件分类器

Blackhat EU 2016

机器学习分类器

- 30 多种特性
- 超过 1.5 万培训样本
- 线性和简单的非线性算法（随机森林）
- 使用 Weka

深度神经网络分类器

- 1000 多种特性
- 超过 1,500 万培训样本
- 复杂的非线性算法
- 使用 PaddlePaddle

深度学习在 CGC 中的使用



使用深度学习的 CGC 团队的数量: 0

为什么不用呢? (我们的回答)

- 没有能够定义成分类问题

- 没有丰富的数据集 (用于培训)



Will Humans or Bots Rule Cybersecurity? The Answer Is Yes

“还有其他类型的人工智能，研究不局限于统计学习。”

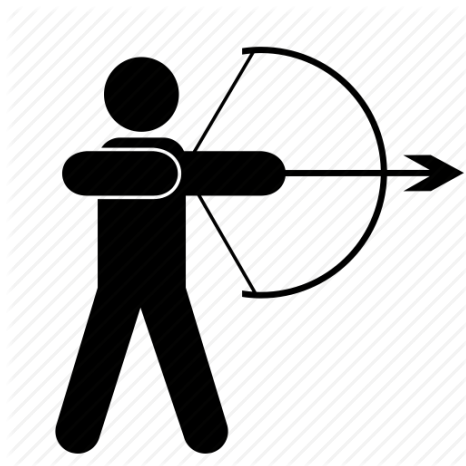
DARPA 信息创新部门负责人 John Launchbury 表示。

<https://www.wired.com/2016/08/will-humans-bots-rule-cybersecurity-answer-yes/>

AI & FUZZING

Fuzzing 二进制程序（在CGC场景下）

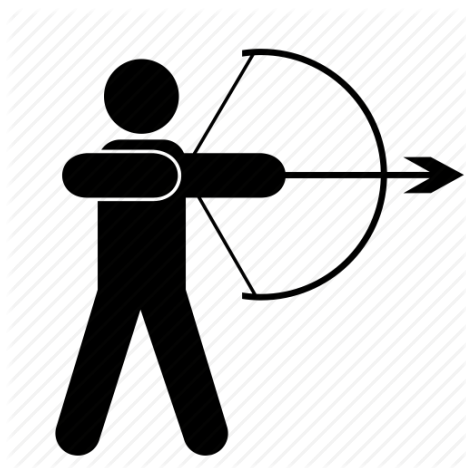
如何生成一个让给定二进制程序“崩溃”的输入？



'\x06\x1d\x0c\x08%7\$s\n\x1f\x1d\x0c\x08%7\$s\n'

Fuzzing 二进制程序（在CGC场景下）

如何生成一个让给定二进制程序“崩溃”的输入？



'\x06\x1d\x0c\x08%7\$s\n\x1f\x1d\x0c\x08%7\$s\n'

没有“大”数据供挖掘！首先需要找到生成输入的方法

Dumb Fuzzing

生成随机输入突变并运行目标

例如 *afl-fuzz -n*

Dumb Fuzzing

生成随机输入突变并运行目标

例如 *afl-fuzz -n*



人工 Fuzzing

人类如何生成新的“有趣的”输入？



GCov

```
public boolean addAll(int index, Collection c) {  
    if(c.isEmpty()) {  
        return false;  
    } else if( size == index || size == 0) {  
        return addAll(c);  
    } else {  
        Listable succ = getListableAt(index);  
        Listable pred = (null == succ) ? null : succ.prev();  
        Iterator it = c.iterator();  
        while(it.hasNext()) {  
            pred = insertListable(pred,succ,it.next());  
        }  
        return true;  
    }  
}
```



人工 Fuzzing

人类如何生成新的“有趣的”输入？

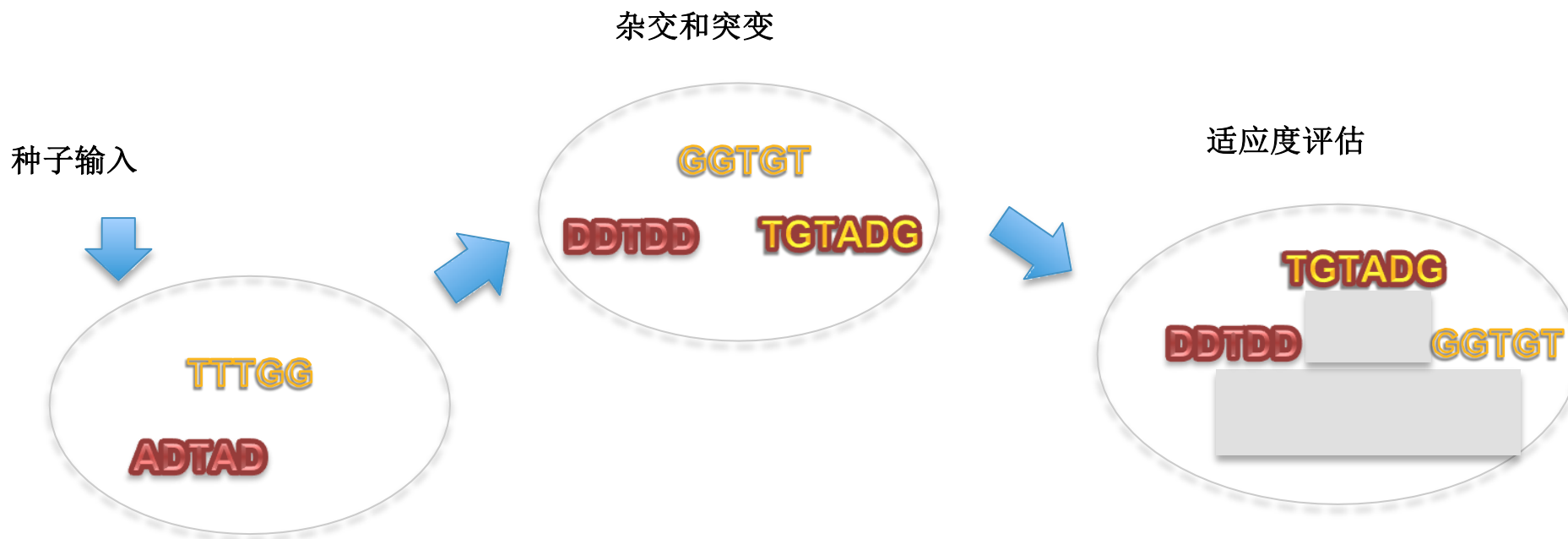


跟踪和观察覆盖范围，然后生成可到达未测试代码的输入



人工智能的演变分支

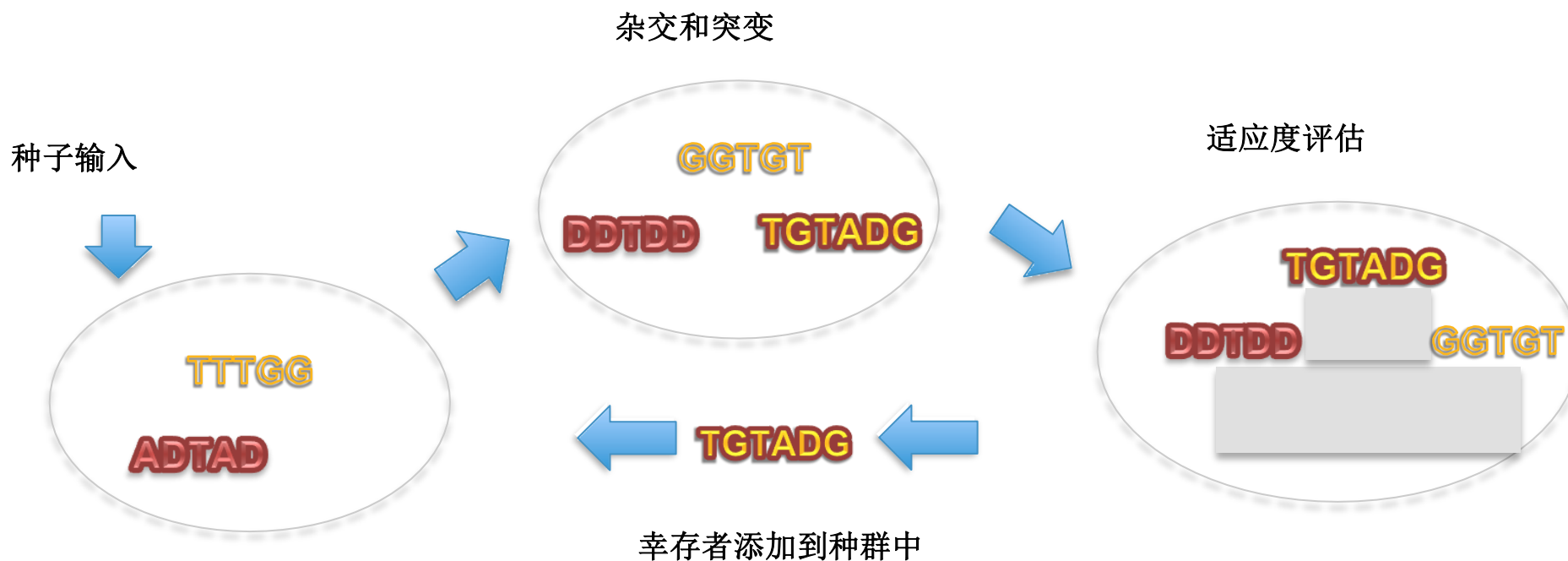
遗传算法 -- 基于自然选择过程





人工智能的演变分支

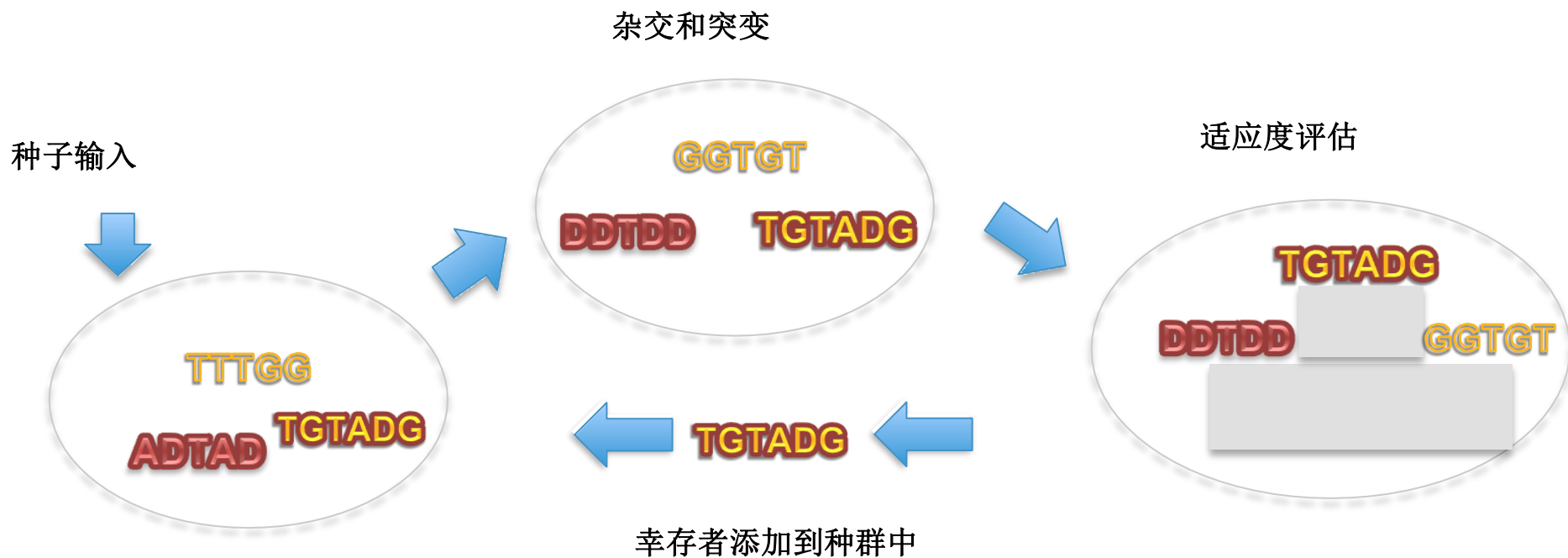
遗传算法 -- 基于自然选择过程





人工智能的演变分支

遗传算法 -- 基于自然选择过程



AFL - Fuzzing 对遗传算法的使用

通过种子输入变换生成新的输入

位翻转/字节翻转/整数运算.....

从每次测试运行中获取反馈

测试控制流路径覆盖

进行适应性测试和突变选择

如果触发新的边缘覆盖 ➔ 添加到种子种群

AFL 插桩

来自 *dwarfdump/print_abbrevs.c*

... ..

328 if (abbrev_code > 0)

{ 329 check_abbrev_num_sequence(...,
330 ...);

331 while (abbrev_code >= abbrev_array_size) {

332 Dwarf_Unsigned old_size = abbrev_array_size;
333 size_t addl_size_bytes = old_size *
... ..

BB1

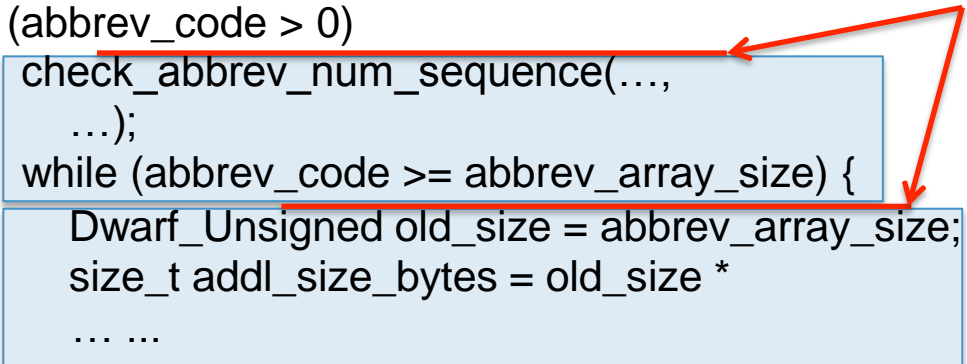
BB2

通过更新 64KB 共享内存区域 (*afl_area_ptr* []) 收集覆盖信息

AFL 插桩

来自 *dwarfdump/print_abbrevs.c*

```
... ..
328 if (abbrev_code > 0)
{ 329     check_abbrev_num_sequence(...,
330         ...);
331     while (abbrev_code >= abbrev_array_size) {
332         Dwarf_Unsigned old_size = abbrev_array_size;
333         size_t addl_size_bytes = old_size *
... ..
```



afl_area_ptr [cur_loc ^ prev_loc]++;

通过更新 **64KB** 共享内存区域 (**afl_area_ptr []**) 收集覆盖信息

AFL 的成功

快速和可靠的 **Fuzzing**

测试开销低，使用简单

漏洞发现实例

Bind、PuTTY、tcpdump、ffmpeg、GnuTLS、libtiff、libpng.....更多信息请查看 AFL 网站 (<http://lcamtuf.coredump.cx/afl/>)

被广泛使用

大多 2016 CGC 获奖团队都使用了这种插桩技术

AFL 面临的挑战

... ..

```
if (input == 0xDEADBEEF)
    crash();
else
    good();
```

... ..



通过输入突变触发崩溃
分支的几率非常低！

AFL 面临的挑战（续）

来自 Regehr 教授的博客

<http://blog.regehr.org/archives/1238>



What afl-fuzz Is Bad At

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main(void) {
5      char input[32];
6      if (fgets(input, 32, stdin)) {
7          long n = strtol(input, 0, 10);
8          printf("%ld\n", 3 / (n + 1000000));
9      }
10     return 0;
11 }
```

AFL 面临的挑战（续）

来自 Regehr 教授的博客

<http://blog.regehr.org/archives/1238>



What afl-fuzz Is Bad At

对于 n 的所有数值，只有
当 $n == -1000000$ 时
才会发生 DIVZ 错误

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main(void) {
5      char input[32];
6      if (fgets(input, 32, stdin)) {
7          long n = strtol(input, 0, 10);
8          printf("%ld\n", 3 / (n + 1000000));
9      }
10     return 0;
11 }
```

在 CGC 中我们为 AFL 添加的增强功能

两项增强功能

1. 采用了 **QEMU** 增强插桩的 **AFL (Qai)**

2. 采用了符号执行协助的 **AFL**

AFL 与人工选择的对决

... ..

```
ival = func(input);
```

```
if (ival == 0xDEADBEEF)
```

```
    crash();
```

```
else
```

```
    good();
```

... ..

考虑两种引起 `ival` 等于下列两个值的输入

1) 0x12345678

2) 0x0000BEEF

afl-fuzz 获得相同的覆盖，

但是，人会怎么做？

AFL 与人工选择的对决

... ..

```
ival = func(input);
```

```
if (ival == 0xDEADBEEF)
```

```
    crash();
```

```
else
```

```
    good();
```

... ..

考虑两种引起 `ival` 等于下列两个值的输入

1) 0x12345678

2) 0x0000BEEF

afl-fuzz 获得相同的覆盖，

但是，人会怎么做？

输入 2 可能是促进进一步突变的更好的候选值！

通过代码转换提高几率

....

```
if (ival == 0xDEADBEEF)
    crash();
```



....

```
p = (char *)&ival;
if (p[0] == 0xEF) if
    (p[1] == 0xBE)
        if (p[2] == 0xAD)
            if (p[3] == 0xDE)
                crash();
```

选出准确输入的几率

$\sim 1/2^{32}$



$\sim 1/2^{10}$

代码转换

laf-intel 最近公布了具体思路

分割-比较-通过、比较-转换-通过等

<https://lafintel.wordpress.com/2016/08/15/circumventing-fuzzing-roadblocks-with-compiler-transformations/>

集成到 AFL-llvm 模式中

问题：

需要重新编译目标程序，但是我们只有二进制程序。



当我们只有二进制程序时，如何为 **AFL** 提供“部分匹配”反馈？

插桩粒度

转化块级别的 AFL QEMU 插桩

```
00000000040066d: <main>:
40066d: 55          push    %rbp
40066e: 48 89 e5    mov     %rsp,%rbp
400671: 53          push    %rbx
400672: 48 83 ec 48 sub     $0x48,%rsp
400676: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
40067d: 00 00
40067f: 48 89 45 e8 mov     %rax,-0x18(%rbp)
400683: 31 c0       xor     %eax,%eax
400685: 48 8b 15 cc 09 20 00 mov     0x2009cc(%rip),%rdx
40068c: 48 8d 45 c0 lea     -0x40(%rbp),%rax
400690: be 20 00 00 00 mov     $0x20,%esi
400695: 48 89 c7    mov     %rax,%rdi
400698: e8 b3 fe ff ff callq   400550 <fgets@plt>
40069d: 48 85 c0    test    %rax,%rax
4006a0: 74 41       je      4006e3 <main+0x76>
4006a2: 48 8d 45 c0 lea     -0x40(%rbp),%rax
```

二进制文件



```
---- 0x4006e0
movi_i64 tmp0,$0x4006e5
goto_tb $0x0
movi_i64 tmp3,$0x400530
st_i64 tmp3,env,$0x80
exit_tb $0x7f72cad71ea0
set_label $L0
exit_tb $0x7f72cad71ea3

---- 0x400530
movi_i64 tmp2,$0x601020
qemu_ld_i64 tmp0,tmp2,le,0x0
st_i64 tmp0,env,$0x80
exit_tb $0x0
set_label $L0
exit_tb $0x7f72cad71f13
```

转化块链

afl_area_ptr [position]++;

afl_area_ptr [position]++;

我们需要运行时指令级别的插桩

TCG 指令级别的转化



I386

```
80484d7:  
cmp $0xdeadbeef,%eax
```

TCG IR

```
---- 0x80484d7  
mov_i32 tmp1,$0xdeafbeef  
mov_i32 tmp0,eax  
mov_i32 cc_src,tmp1  
mov_i32 loc10,tmp0  
sub_i32 cc_dst,tmp0,tmp1
```

x86_64

```
...  
mov $0xdeedbeef,%ebp  
mov %ebp,0x28(%r14)  
mov $0x10,%ebx  
mov %ebx,0x34(%r14)  
test %ebp,%ebp
```

TCG 辅助函数！



I386

```
4006d0:    idiv  %rbx
```

TCG IR

```
---- 0x4006d0
mov_i64 tmp0,rbx movi_i64
tmp3,$0x4006d0 st_i64
tmp3,env,$0x80
call idivq_EAX,$0x0,$0,env,tmp0
```

x86_64

```
...
mov  %rbx,0x10(%r14)
movq $0x4006d0,0x80(%r14)
mov  %r14,%rdi
mov  %rbp,%rsi
callq 0x7f3037453600
```

到 TCG 分支操作的插桩



I386

```
80484dc:jne      80484fe
```

TCG IR

```
---- 0x80484dc  
movi_i32 cc_op,$0x10  
discard loc10  
movi_i32 tmp11,$0x0  
brcond_i32 cc_dst,tmp11,ne,$L1
```

调用 **brcond** 辅助函数

if (partial match)
alf_area_ptr[pos]++

采用 QEMU 增强插桩的 AFL

对 **TCG** 分支操作生成的修改


tcg_gen_brcond_i32()

tcg_gen_brcond_i64()

...

添加一个辅助函数来更新覆盖反馈

剩下的问题是：更新哪些分支边缘？

```
cur_location = (block_address >> 4) ^ (block_address << 8);  
afl_area_ptr[cur_loc  prev_loc]++;  
prev_loc = cur_loc >> 1;
```

AFL-Qai 实现

增强的插桩示例（针对 **64** 位分支指令）

```
uint64_t HELPER(brcond_states_i64)(uint64_t cond, uint64_t nztest, uint64_t addr)
{
    uint64_t mask = 0xff00000000000000;
    ...
    while (!(cond&mask) && i<8) {
        mask = mask>>8;
        i++;
    }
    if (i>0) {
        fake_edge = (i+1)*addr % bitmap_size;
        afl_area_ptr[fake_edge]++;
    }
    ...
}
```


采用 QEMU 增强指令的 AFL

为其他指令添加扩充的适应性反馈

例如，增加捕获 DIVZ 错误的几率

```
void helper_idivq_EAX() {  
    // if the divisor is close to zero  
    afl_area_ptr[addr*matching_factor]++;  
}
```

采用 QEMU 增强插桩的 AFL

扩充的反馈

返回更为精细的适应度信息，即使是面向相同的边缘覆盖

例如，更新 **afl** 共享的内存以进行部分匹配

支持多种二进制平台

在 QEMU TCG 识别实施，适用于多个架构

american fuzzy lop (fast) 2.33b (rock)

process timing		overall results
run time : 5 days, 21 hrs, 8 min, 53 sec		cycles done : 130k
last new path : none yet (odd, check syntax!)		total paths : 1
last uniq crash : none seen yet		uniq crashes : 0
last uniq hang : 5 days, 9 hrs, 4 min, 11 sec		uniq hangs : 1
cycle progress		map coverage
now processing : 0.130433 (0.00%)		map density : 0.04% / 0.04%
paths timed out : 0 (0.00%)		count coverage : 1.00 bits/tuple
stage progress		findings in depth
now trying : havoc		avored paths : 1 (100.00%)
stage execs : 1660/4096 (40.53%)		new edges on : 1 (100.00%)
total execs : 534M		total crashes : 0 (0 unique)
exec speed : 1045/sec		total hangs : 3 (1 unique)
fuzzing strategy yields		path geometry
bit flips : 0/32, 0/31, 0/29		levels : 1
byte flips : 0/4, 0/3, 0/1		pending : 0
arithmetics : 0/224, 0/0, 0/0		pend fav : 0
known ints : 0/20, 0/84, 0/44		own finds : 0
dictionary : 0/0, 0/0, 0/0		imported : n/a
havoc : 0/534M, 0/0		stability : 100.00%
trim : 55.56%/2, 0.00%		

[cpu002: 86%]

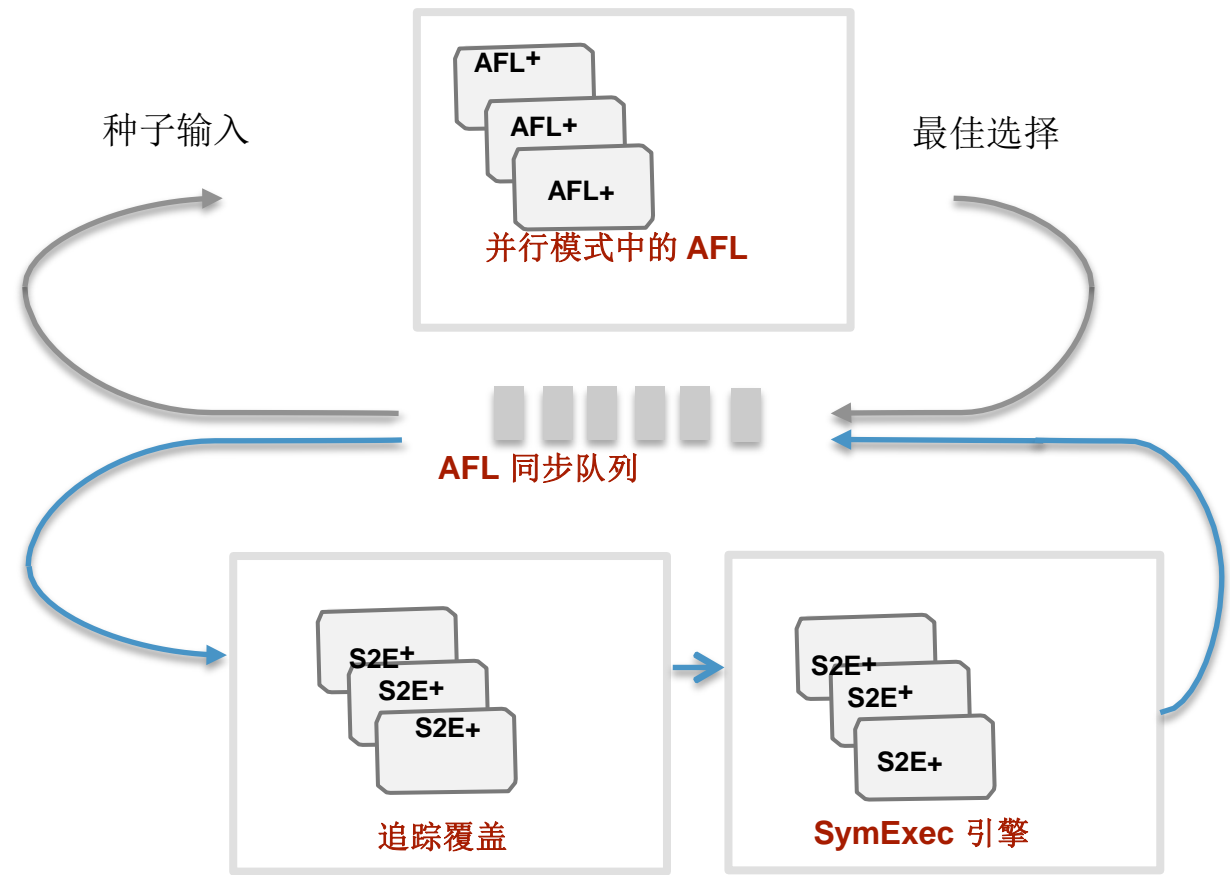
american fuzzy lop 2.35b (rock)

process timing		overall results
run time : 0 days, 0 hrs, 6 min, 43 sec		cycles done : 317
last new path : 0 days, 0 hrs, 3 min, 57 sec		total paths : 4
last uniq crash : 0 days, 0 hrs, 0 min, 52 sec		uniq crashes : 1
last uniq hang : none seen yet		uniq hangs : 0
cycle progress		map coverage
now processing : 1 (25.00%)		map density : 0.28% / 0.29%
paths timed out : 0 (0.00%)		count coverage : 1.00 bits/tuple
stage progress		findings in depth
now trying : havoc		avored paths : 4 (100.00%)
stage execs : 114/256 (44.53%)		new edges on : 4 (100.00%)
total execs : 779k		total crashes : 1 (1 unique)
exec speed : 1922/sec		total hangs : 0 (0 unique)
fuzzing strategy yields		path geometry
bit flips : 0/128, 0/124, 0/116		levels : 2
byte flips : 0/16, 0/12, 0/4		pending : 0
arithmetics : 0/894, 0/204, 0/0		pend fav : 0
known ints : 0/78, 0/316, 0/176		own finds : 3
dictionary : 0/0, 0/0, 0/0		imported : n/a
havoc : 2/301k, 2/476k		stability : 100.00%
trim : 42.86%/3, 0.00%		

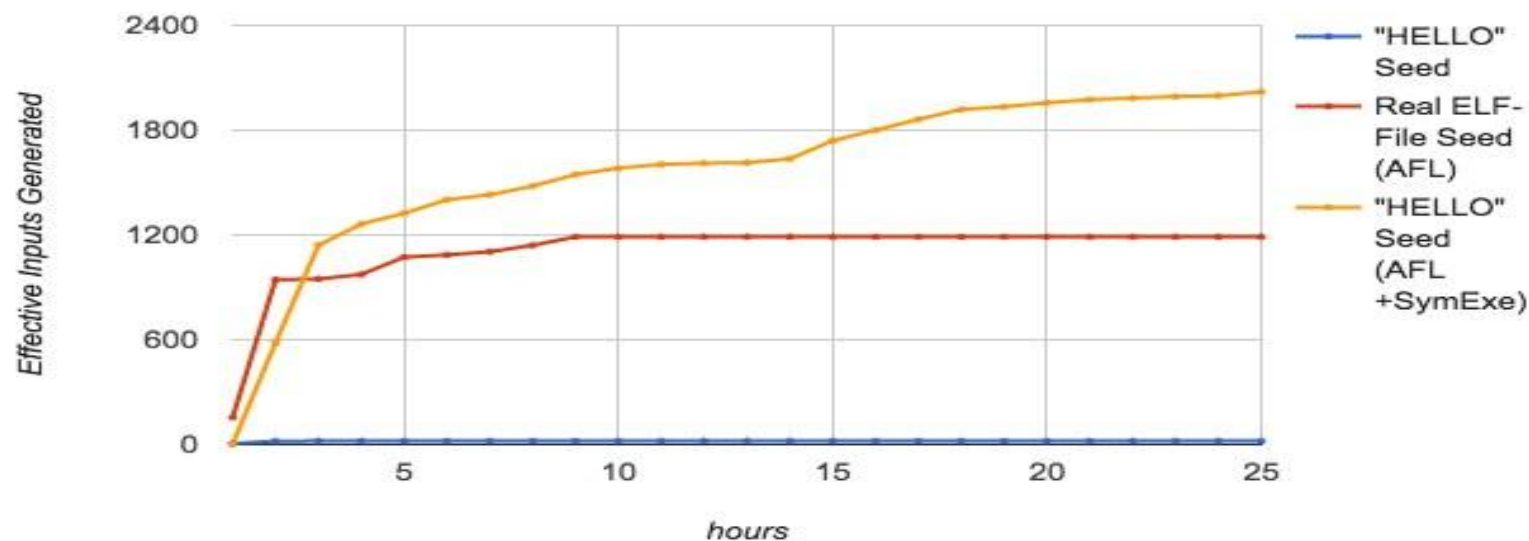
[cpu002: 86%]

AFL + SymEXC

采用了符号执行的 AFL



采用 SymExec Aid 的 AFL



对照 *readelf* 程序运行 AFL （单个 AFL 实例）

总结

关于深度学习和 CGC

1. 在 2016 年 CGC 中，没有团队使用深度学习
2. 需要制定分类问题，并且需要数据

在漏洞发现中更多地运用人工智能的潜力

1. 挖掘漏洞模式

还没有采用深度学习（机遇？）

2. 人工辅助 fuzzing

如何充分运用人类洞察？

总结（续）

*我们提升 **AFL Fuzzing** 智能的方法*

1. 使用更精细覆盖插桩的 Fuzzing
2. 采用符号执行协助的 Fuzzing

致谢

Disekt 团队

Michael Contreras、Rober Lee Harrison、Yeongjin Jang、Taesoo Kim、
Byoung Young Lee、Chengyu Song、Kevin Warrick、Insu Yun

及其他合作者

Manos Antonakakis、Babak Rahbarinia、Roberto Perdisci、Phani
Vadrevu、Qixue Xiao、Guodong Zhu

问答

kangli.ctf@gmail.com