2019-2020学年秋季学期

漏洞利用与攻防实践

Exploiting Software Vulnerability-Techniques and Practice

报告：张志杰 侯贵洋

演示：董国超 申卓祥

漏洞利用与攻防实践

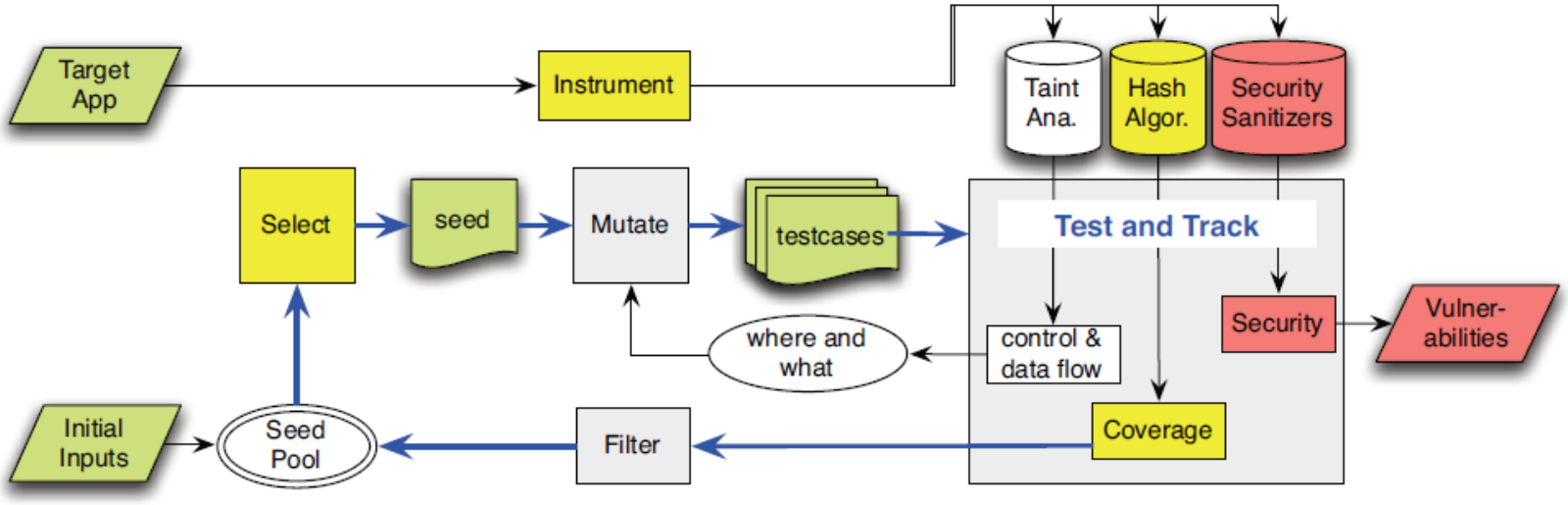
Exploiting Software Vulnerability-Techniques and Practice

[第五次课] AFL 实践

概 要

- 回顾AFL
- CollAFL：Path Sensitive Fuzzing
- 实验

回顾AFL



AFL流程：

1. 根据预定规则从种子池里选择种子
2. 对输入的种子进行突变以产生测试用例
3. 用这些测试用例对目标应用进行高速测试
4. 用插桩监控程序执行过程，追踪覆盖率和违反安全的情况
5. 如果检测到违反安全的情况，报告弱点
6. 根据代码覆盖率过滤出的测试用例，放入种子池中，回到步骤1

回顾AFL

- 边、块、插桩、代码覆盖率的概念

- **插桩：** 插桩是为了覆盖率而实行的方法

- **代码覆盖率：** 度量代码的覆盖程度的方式，也就是指源代码中的某行代码是否已执行。

- **边界：** 将程序看成一个控制流图（CFG），图的每个节点表示一个基本块，而边就被用来表示在基本块之间的转跳。

- **基本块：** 指一组顺序执行的指令，基本块中第一条指令被执行后，后续的指令也会被全部执行，每个基本块中所有指令的执行次数是相同的

AFL 插桩思路

有源码

1. cur_location = <COMPILE_TIME_RANDOM>;

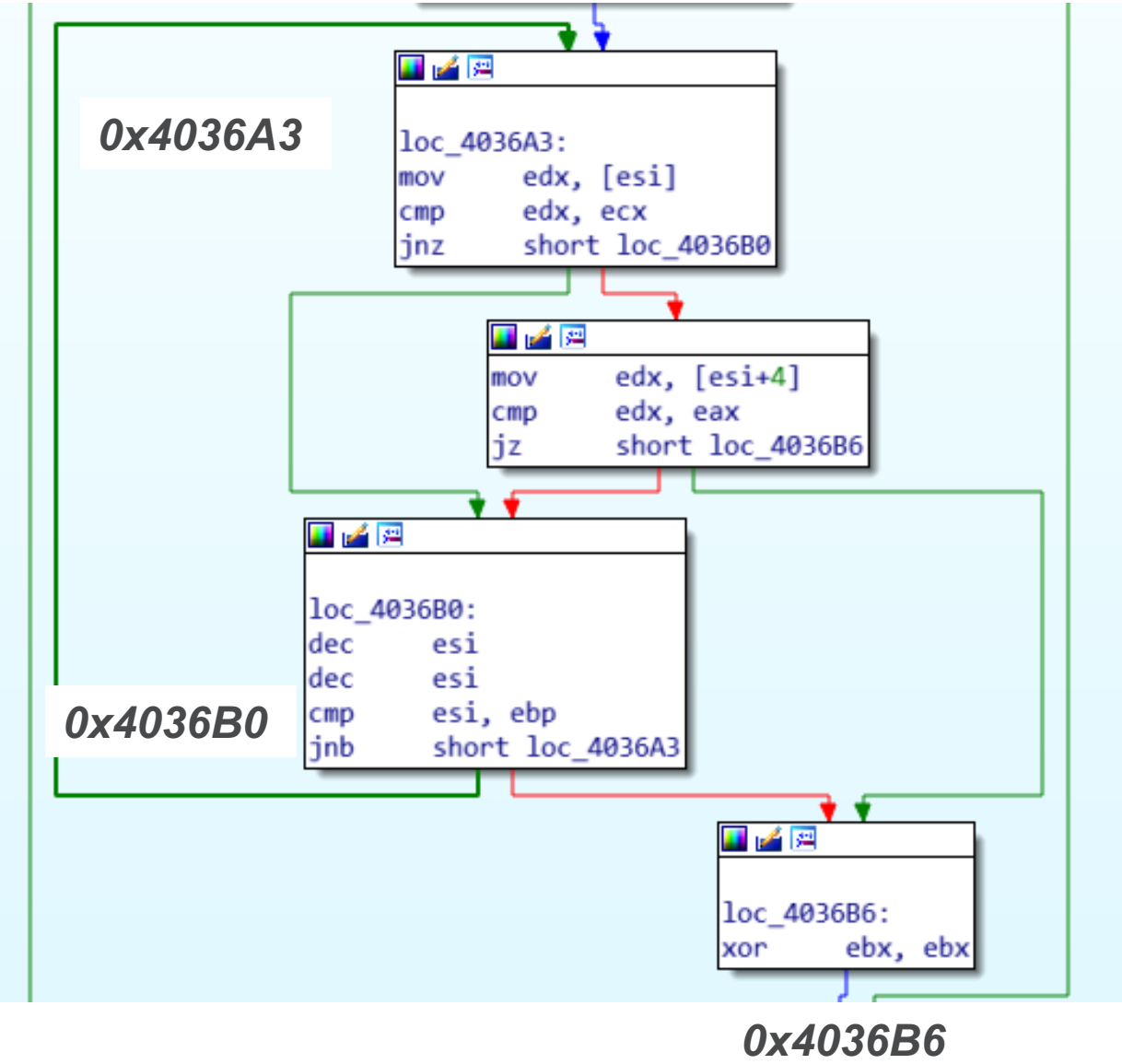
2. shared_mem[cur_location ^ prev_location]++;

3. prev_location = cur_location >> 1;

覆盖率信息如何收集 ？

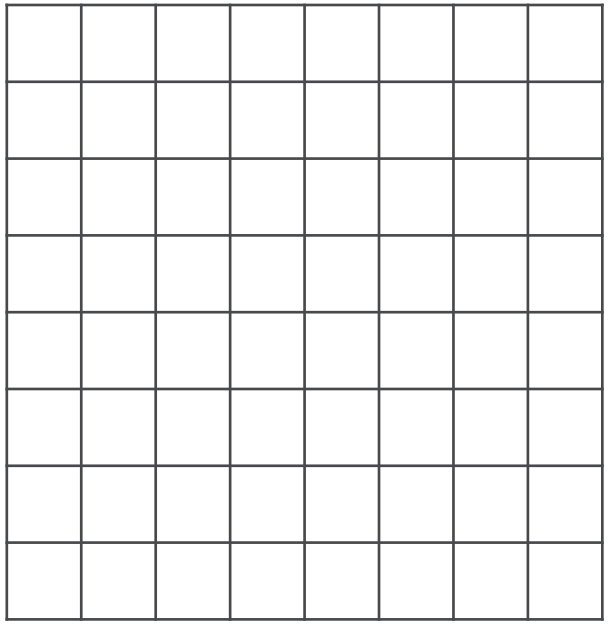
为每一个被覆盖的初始块设定一个随机数：

…
0x4036A3
0x4036B0
0x4036B6
…

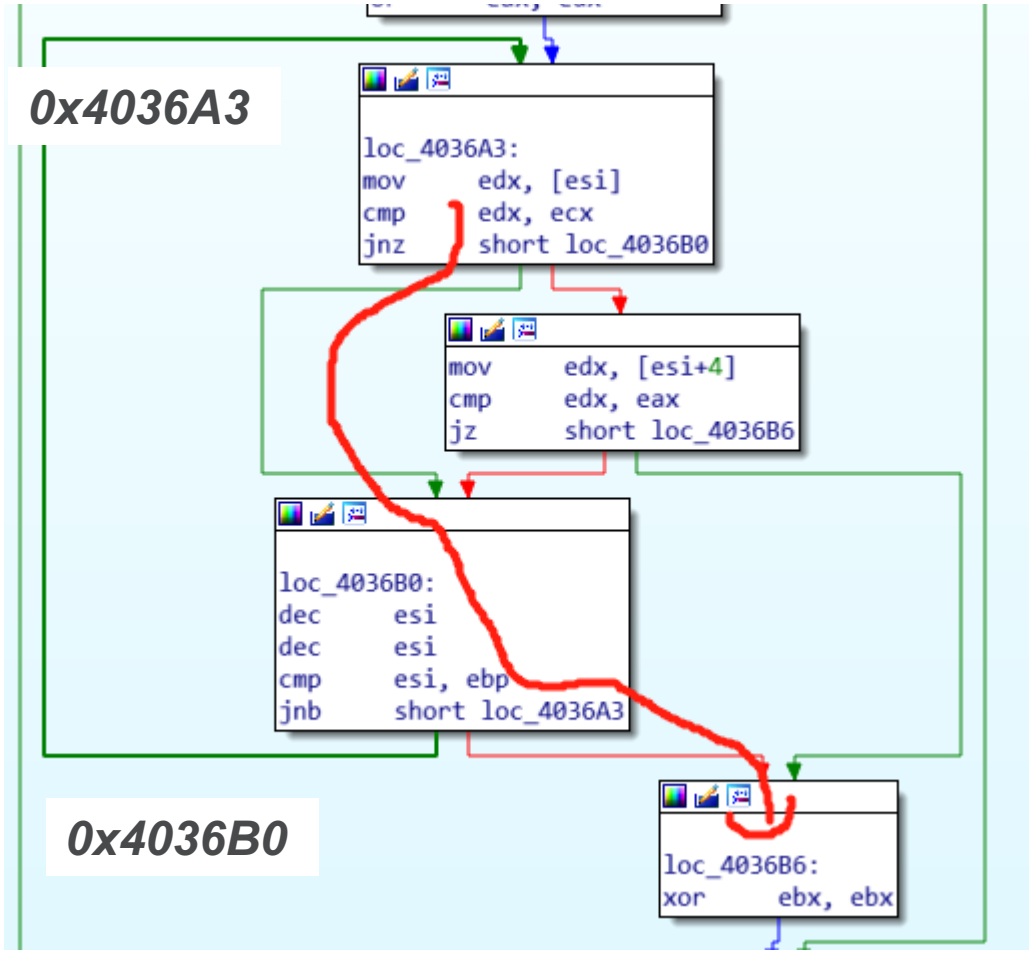


覆盖率信息如何收集 ？

test case 1
…
0x4036A3
0x4036B0
0x4036B6
…

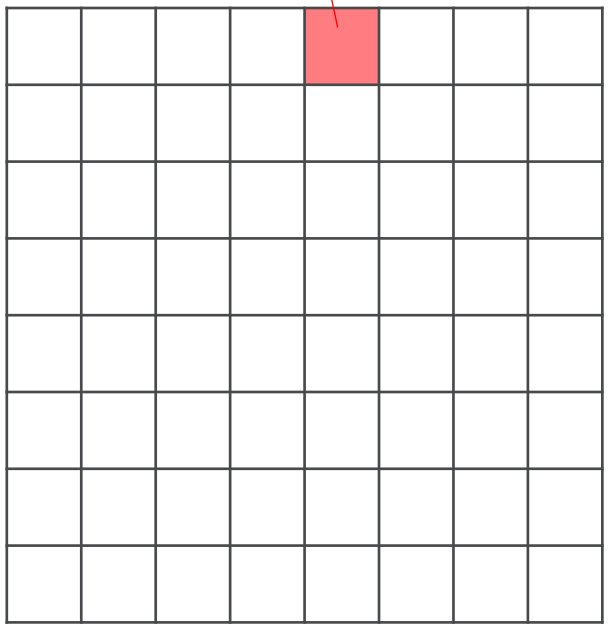


shared_mem[ ]

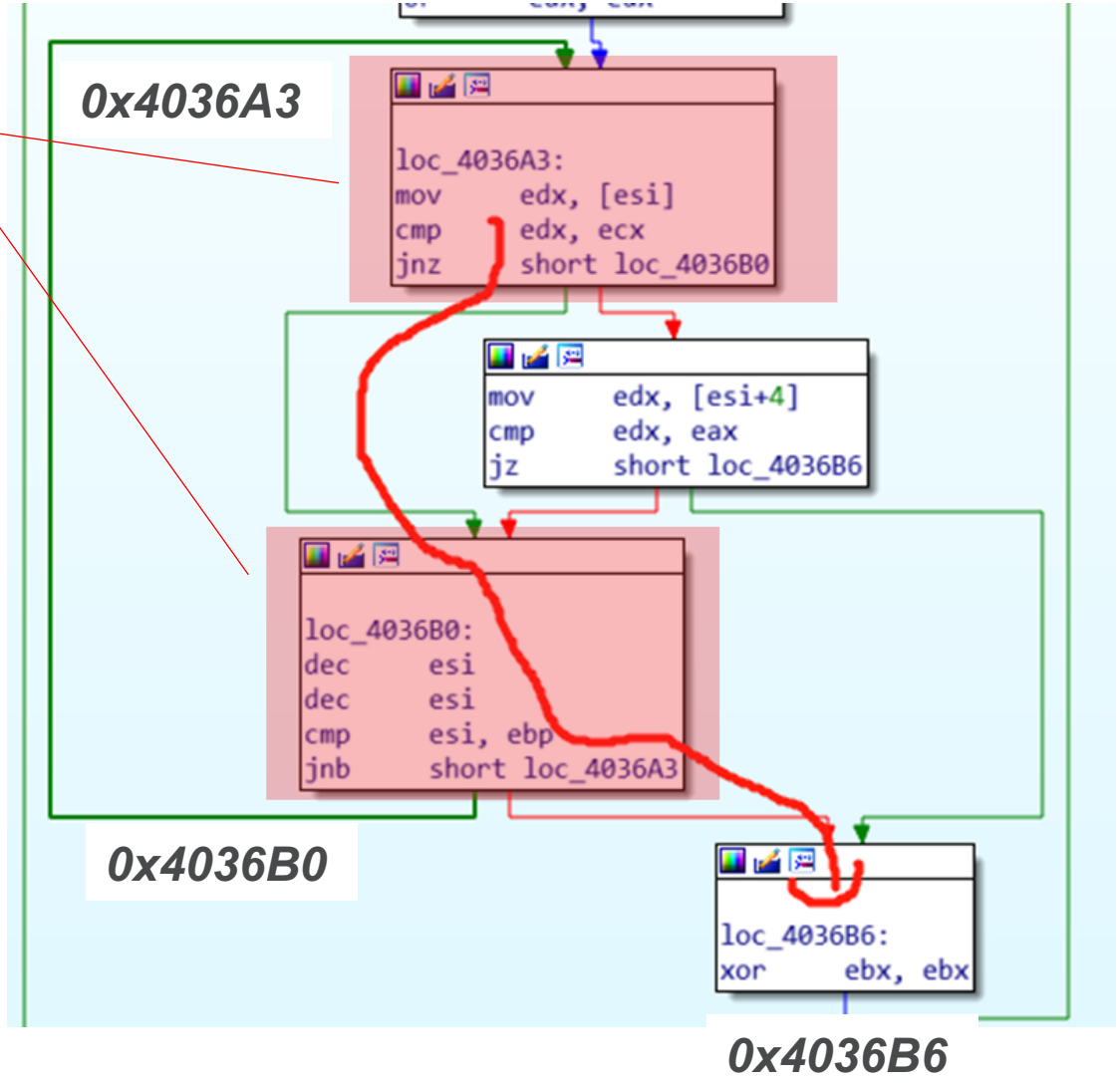


覆盖率信息如何收集 ？

Hash(0x4036A3,0x4036B0)



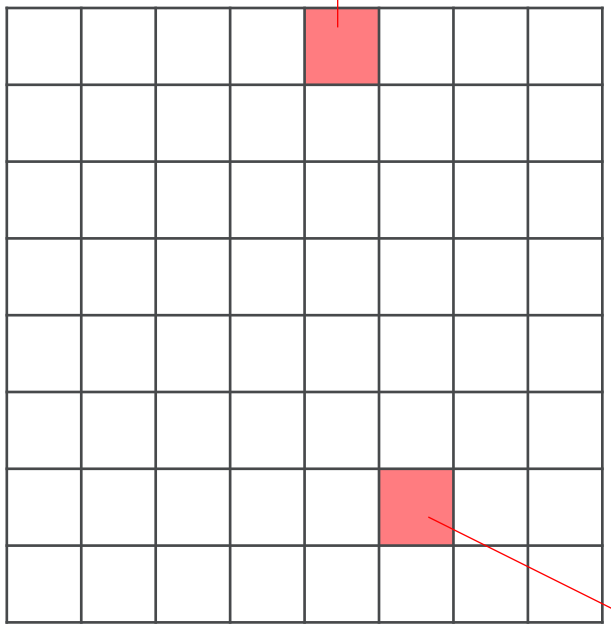
shared_mem[ ]



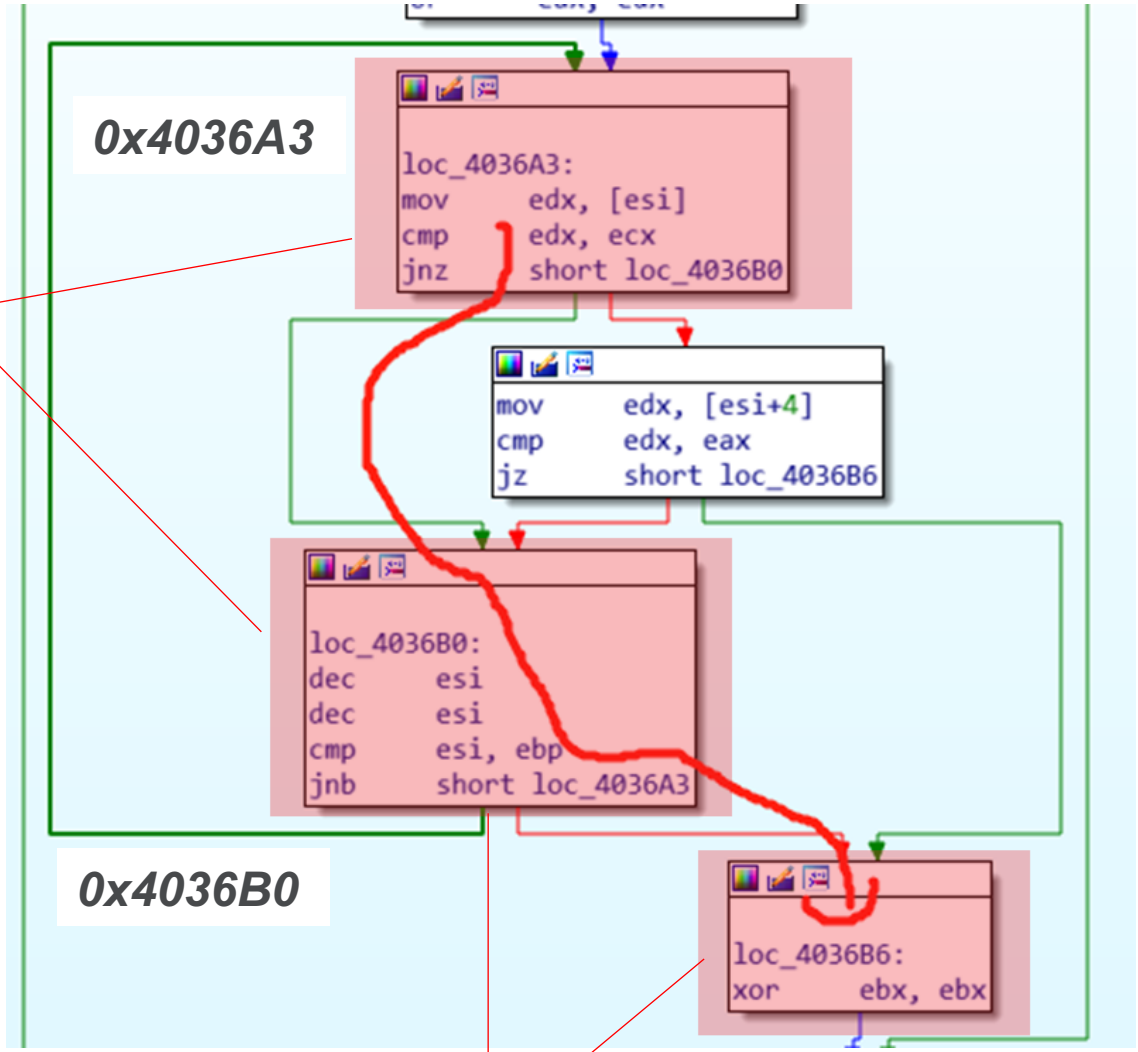
覆盖率信息如何收集 ？

如果shared_mem[ ]的标记更新了，意味着发现了一条新的路径。

Hash(0x4036A3,0x4036B0)



shared_mem[ ]

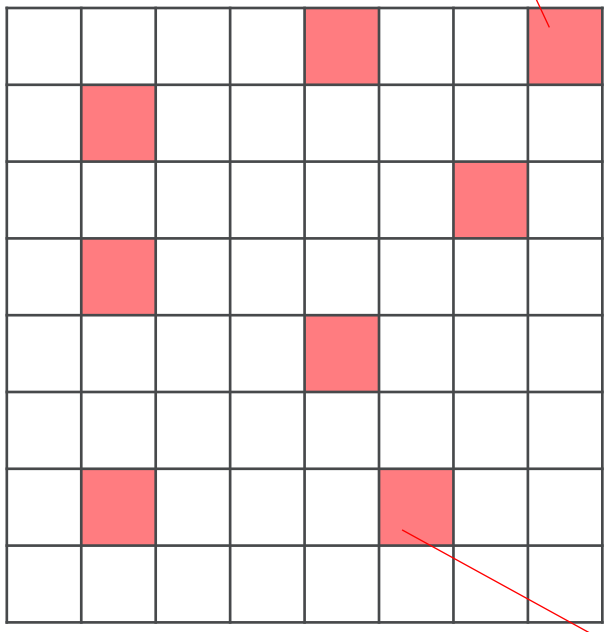


Hash(0x4036B0,0x4036B6)

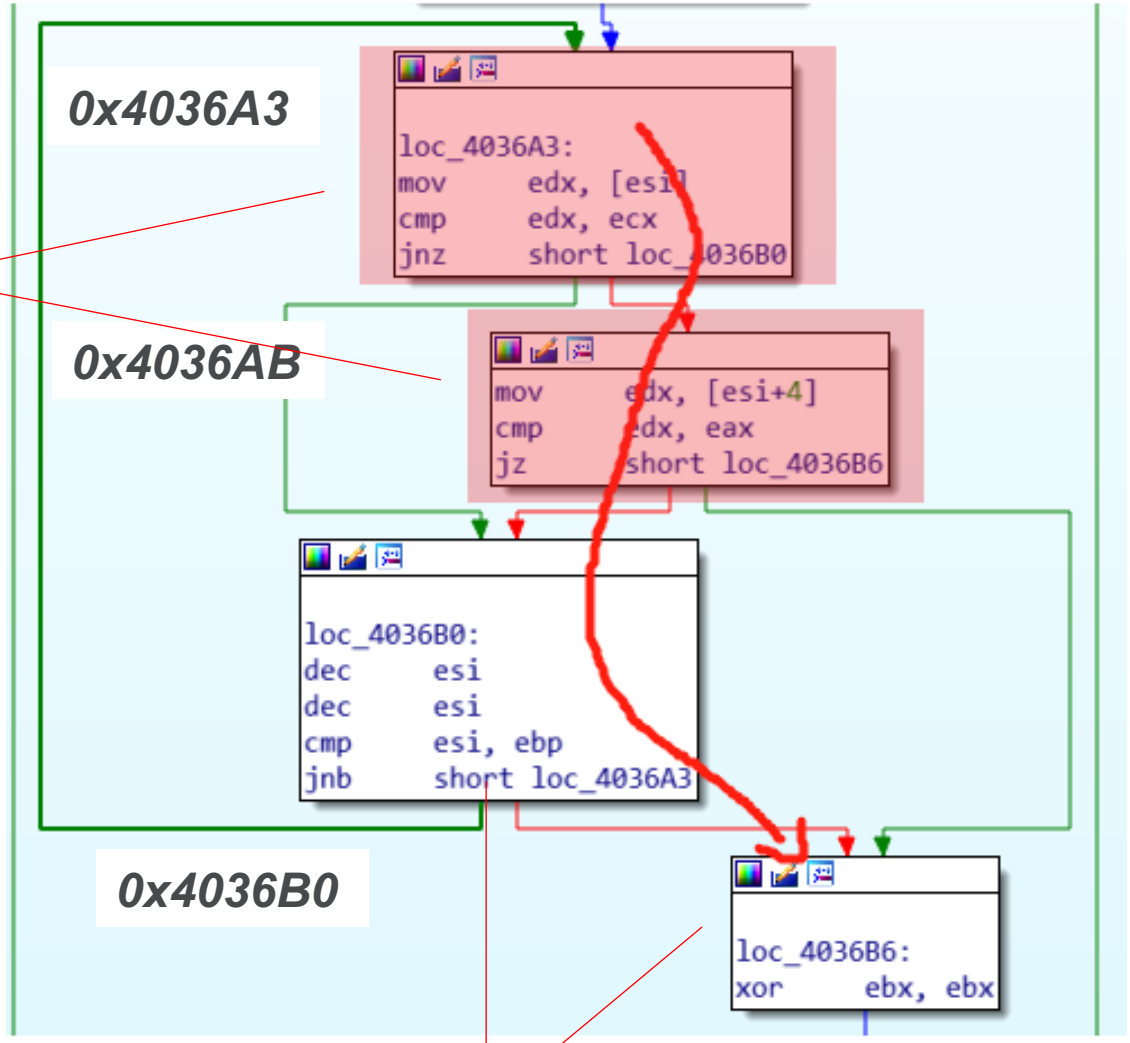
覆盖率信息如何收集 ？

如果shared_mem[ ]的标记没有更新，就说明没有找到新的路径。

Hash(0x4036A3,0x4036AB)



shared_mem[ ]

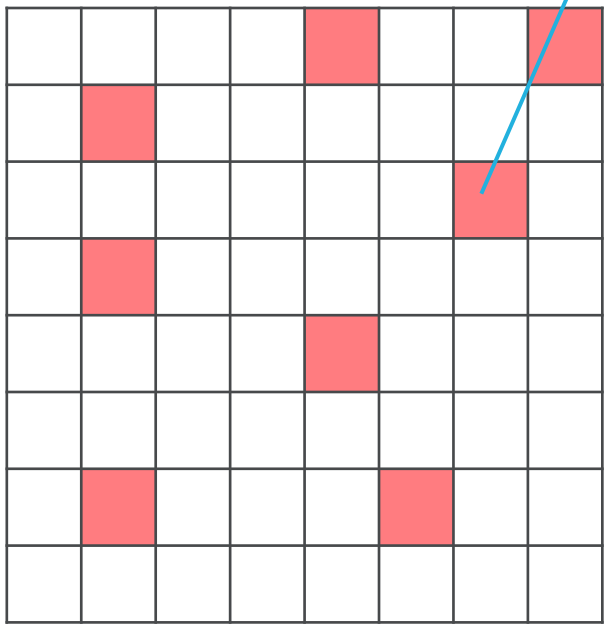


Hash(0x4036B0,0x4036B6)

0x4036B6

哈希冲突是什么 ？

Testcase X :
0x403612->0x4036B2



shared_mem[ ]

Hash(0x4036A3,0x4036AB)

0x403612

某基本块

0x4036A3

```
loc_4036A3:
mov     edx, [esi]
cmp     edx, ecx
jnz     short loc_4036B0
```

0x4036AB

```
mov     edx, [esi+4]
cmp     edx, eax
jz      short loc_4036B6
```

```
loc_4036B0:
dec     esi
dec     esi
cmp     esi, ebp
jnb     short loc_4036A3
```

0x4036B0

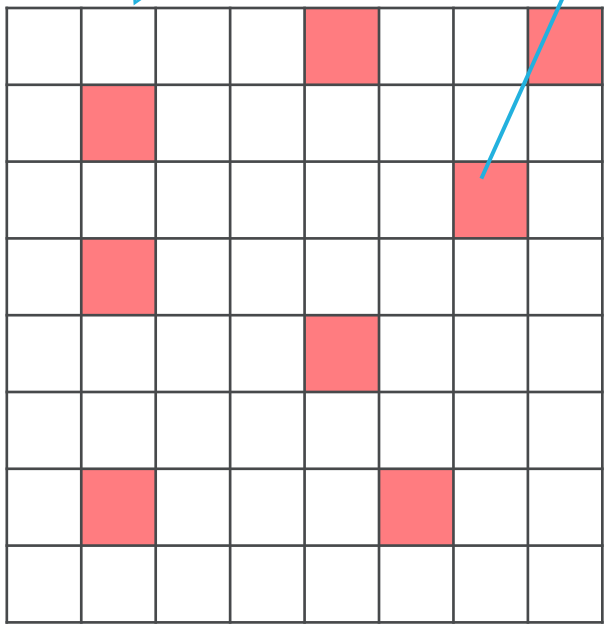
```
loc_4036B6:
xor     ebx, ebx
```

0x4036B6

哈希冲突是什么 ？

Testcase X :
0x403612->0x4036B2

Hash
冲突！

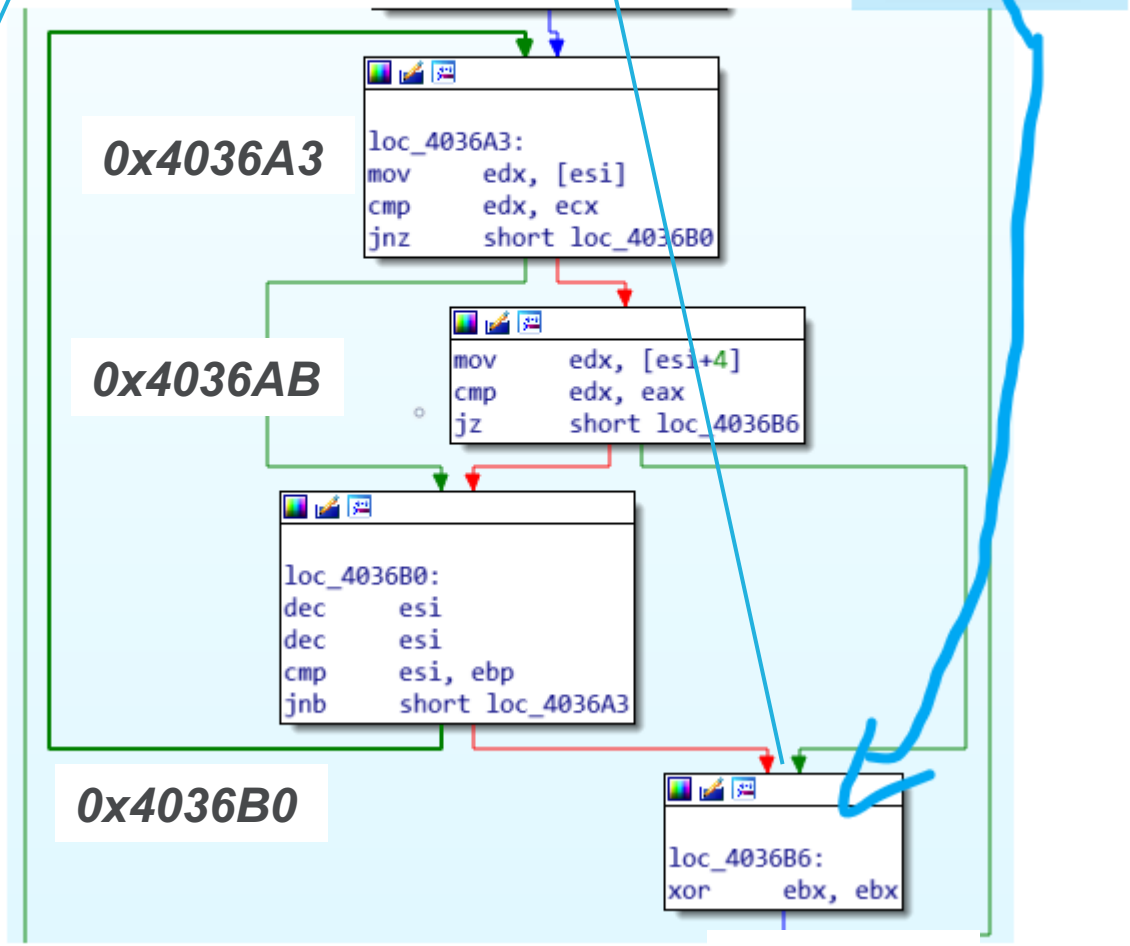


shared_mem[ ]

Hash(0x4036A3,0x4036AB)

0x403612

某基本块



0x4036B6

CollAFL : Path Sensitive Fuzzing

背景

- 准确的覆盖率信息可以提高发现漏洞的概率，但同时会带来极大的开销。
- 实验表明超过75%的边存在哈希碰撞。
- 由于AFL的成功，覆盖率不准确的后果被忽视了。

CollAFL : Path Sensitive Fuzzing

三个问题

- 为什么覆盖率准确性会影响漏洞挖掘？

CollAFL : Path Sensitive Fuzzing

三个问题

- 为什么覆盖率准确性会影响漏洞挖掘？
- 如何提高覆盖率准确性？

CollAFL : Path Sensitive Fuzzing

三个问题

- 为什么覆盖率准确性会影响漏洞挖掘？
- 如何提高覆盖率准确性？
- 覆盖率敏感的模糊测试性能如何？

CollAFL : Path Sensitive Fuzzing

论文方案思路

- 提高覆盖率准确性
- 改良种子选择策略

CollAFL : Path Sensitive Fuzzing

提高覆盖率准确性

- 前提：bitmap size > edge size
- Block with multiple precedent ，计算Fmul

$$Fmul(cur, prev) = (cur \gg x) \oplus (prev \gg y) + z$$

- Unsolvable block with multiple precedent：计算Fhash

$$Fhash(cur, prev) : hash\_table\_lookup(cur, prev)$$

- Block with single precedent：计算 Fsingle

$$Fsingle(cur, prev) : c$$

CollAFL : Path Sensitive Fuzzing

- **种子选择策略（针对以下路径突变）**
  - 有很多没有探索过的邻近分支的路径
  - 有很多没有探索过的邻近子节点的路径
  - 有很多内存权限操作的路径

CollAFL : Path Sensitive Fuzzing

结论

- 提出了基于覆盖率敏感的CollAFL
- 发现了24个真实应用的157个安全漏洞，其中95个CVE编号

实验

- 实验目的
  - Bitmap大小与哈希碰撞率的关系
  - 程序规模与哈希碰撞率的关系
- 实验环境（运行时间：11h-14h）
  - 服务器1：ubuntu 16.04 阿里云 1核
  - 服务器2：ubuntu 18.04 阿里云 1核
  - 服务器3：ubuntu 18.04 腾讯云 1核
  - 服务器4：ubuntu 18.04 百度云 1核
- 实验关键
  - 如何修改bitmap大小？
  - 如何计算哈希碰撞率？

实验难点

- 如何修改bitmap大小？
  - Config.h文件中修改
- 如何计算哈希碰撞率？
  - 确定某个测试集，将所有覆盖边的hash记录下来，去找是否有一样的hash
  - 统计覆盖边的数量，如果大于bitmap占用的数量，说明发生了碰撞，如果等于说明没有发生碰撞，于是碰撞率可以定义为：

1- bitmap占用数/覆盖边数目

实验结果

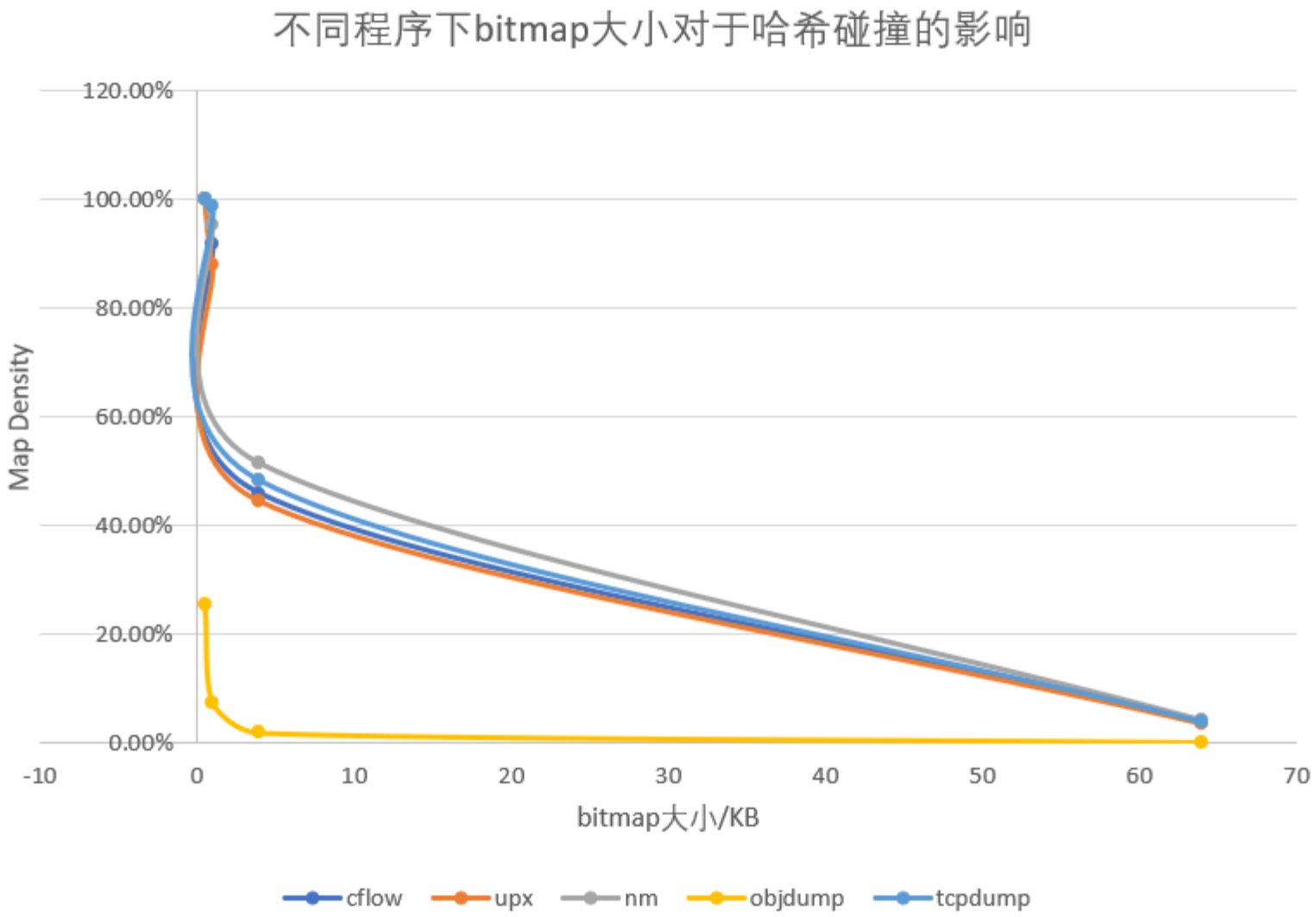
程序规模

cflow	upx	tcpdump	nm	objdump
688KB	1.8M	4.62M	8.72M	11.88M

不同程序规模不同bitmap下的crash/hang

crash/hang	0.5KB	1KB	4KB	64KB
cflow	0/27	61/23	0/102	10/86
upx	4/0	11/0	32/4	5/0
tcpdump	0	0	0	0
nm	0	0	0	0
objdump	0	0	0	0

实验结果


不同程序下bitmap大小对于哈希碰撞的影响
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Map Density
-10
0
10
20
30
40
50
60
70
bitmap大小/KB
cflow
upx
nm
objdump
tcpdump

实验结果

不同程序规模不同bitmap下的总路径

	0.5KB	1KB	4KB	64KB
cflow	589	1459	1906	1730
upx	246	323	461	421
tcpdump	390	1111	914	900
nm	392	1006	1110	1060
objdump	1	1	1	1

实验结论

- Bitmap越大哈希碰撞率越小
- 程序规模越大，在保证相同哈希冲突的情况下，所需要的bitmap也越大

反思

- 错误估计了任务的难度
- 没有意料到执行一次AFL所需要的时间
- 应当在初期测试几个数据，了解大概数据分布，进而合理选择数据

谢谢大家！

参考资料

- CollAFL：http://chao.100871.net/papers/oakland18.pdf
- AFL 漏洞挖掘技术漫谈（一）：https://paper.seebug.org/841/
- AFL 漏洞挖掘技术漫谈（二）：https://paper.seebug.org/842/
- AFL内部实现细节小记：http://rk700.github.io/2017/12/28/afl-internals/
- AFL‘s Blindspot and How to Resist AFL Fuzzing for Arbitrary ELF Binaries – Black Hat 2018 ：https://www.youtube.com/watch?v=pfLpPHN0SjM&list=PLH15HpR5qRsVAXGmSVfjWrGtGLJjlJuGe&index=78&t=0s
- 初探AFL：https://xz.aliyun.com/t/4314