# AFLoW

## 一个有关移植afl到Windows上的故事

主讲：冯震

# 关于作者

冯震 (tinker)

2015.3 上交*毕*业

2015.4 加入k33nteam

@work, c++, python, security ...

@home, soccer, novel, game ...

# 目录

背景: AFL介绍

剧情: AFL on Windows

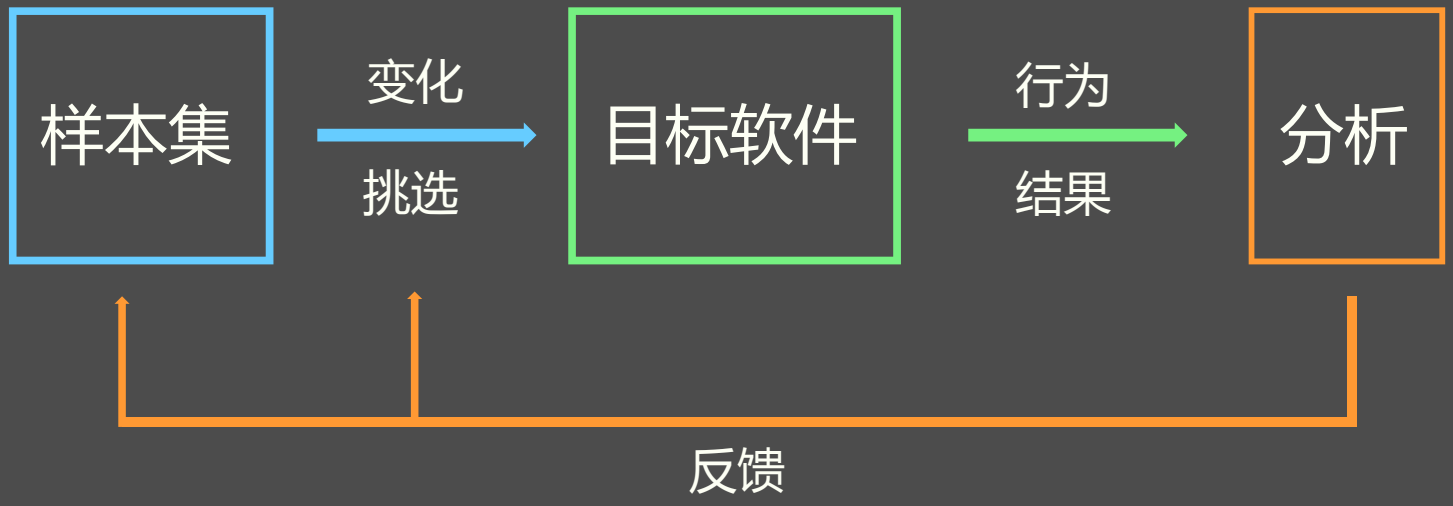
- fork server
- coverage
- genetic algorithm
- bugs

结局: to be better, man

# fuzzing

尽管思想/理念比较简单，却愈发流行，因为非常有效。

dumb, smart/non-dumb, heuristic/feedback, …



# 2014年11月，一个叫做AFL的fuzzer出生了。

AFL is short for American Fuzzy Lop

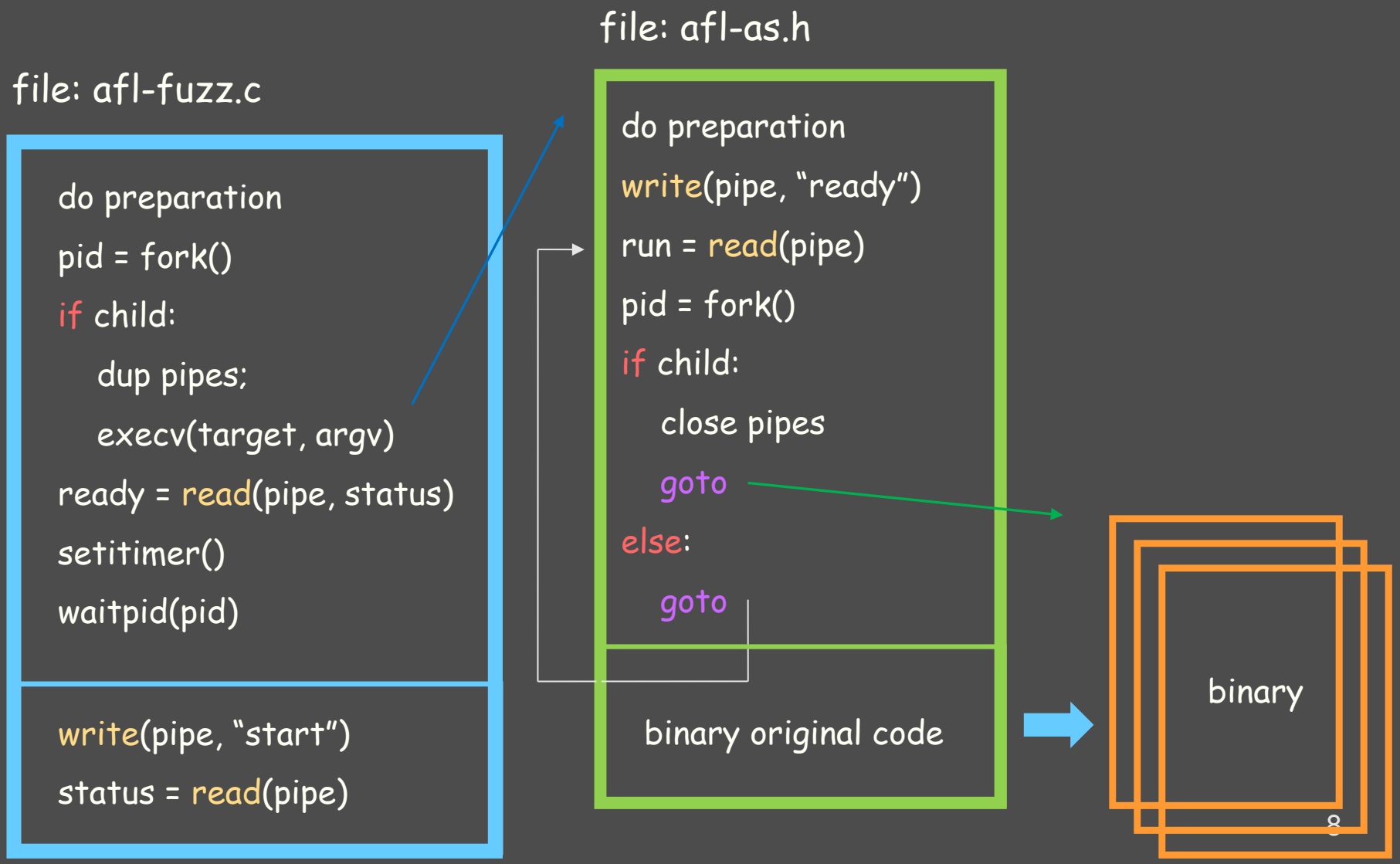
自出生，AFL便集万千宠爱于一身。

fork server

coverage

fuzzing strategy

# fork server


file: afl-fuzz.c
do preparation
pid = fork()
if child:
dup pipes;
execv(target, argv)
ready = read(pipe, status)
setitimer()
waitpid(pid)
write(pipe, "start")
status = read(pipe)
file: afl-as.h
do preparation
write(pipe, "ready")
run = read(pipe)
pid = fork()
if child:
close pipes
goto
else:
goto
binary original code
binary

# fork server

- 非常依赖fork
- 编译时插桩(源码)
- 进程间通信(PIPE)

# coverage

```
curr_location = <COMPILE_TIME_RANDOM>
shared_mem[ curr_location ^ prev_location ] ++
prev_location = curr_location >> 1
```

basic block: A, B, C

A -> B -> C vs A -> C -> B

A -> A vs B -> B

A -> B vs B -> A

A ^ A = B ^ B = 0

A ^ B = B ^ A

# fuzzing strategy

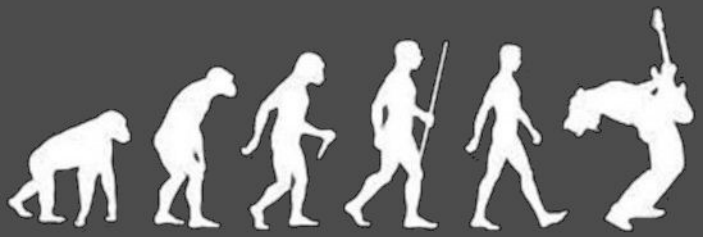
## 遗传算法

要素：

- 个体/种群
- 竞争/适者生存
- 繁衍/变异

算子：

- 选择
- 交叉
- 变异



初始化种群

定义适应度函数

循环：

选择父母

父母交叉 -> 下一代

变异@下一代

计算适应度@下一代

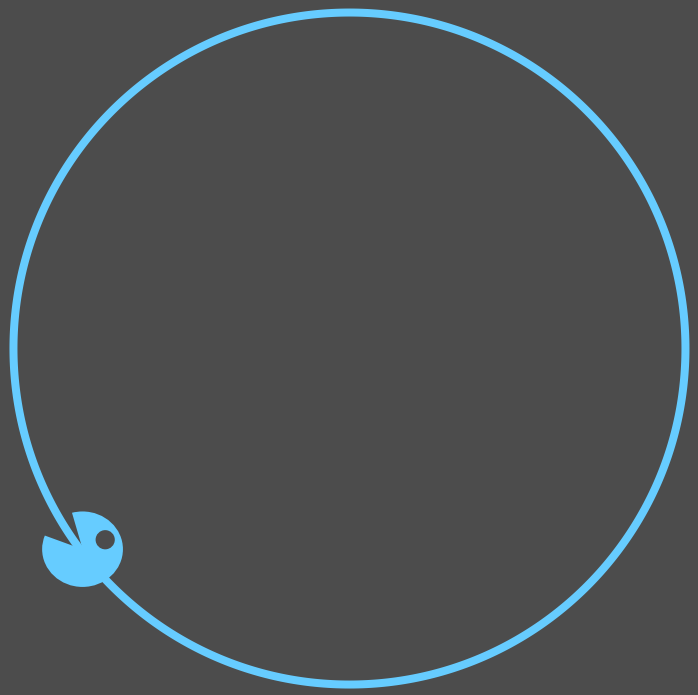
直到：

个体足够好

# fuzzing strategy

file: afl-fuzz.c

```
while True:
    SELECT one from Queue
    SIMPLE BITFLIP
    ARITHMETIC INC/DEC
    INTERESTING VALUES
    DICTIONARY STUFF
    RANDOM HAVOC
    SPLICING
```



# 弱水三千只取一瓢

- new behaviors detecting

  1, 2, 3, 4-7, 8-15, 16-31, …
- input files trimming
- dictionaries
- …

---

更多细节，可以阅读源码或者/docs/technical_details.txt

# AFLoW

## 如果我们要在Windows上实现AFL的拷贝 ...

YES,  ！所有的相关算法都很容易实现或者移植！

BUT,  ？没有fork()？

？没有源码？

OK, let's find the way out.

# core problems

## 为什么需要fork()?

在non-dumb模式中，AFL存在两处代码调用了fork()

- 第一处，只是为了继承管道的文件描述符
- 第二处，则是fork server的核心机制

TEHN, 最根本/抽象的目的又是什么呢？

Performance

# core problems

SO, 关于fork()我们能够做哪些？

方案1：多线程/多进程

MT，资源共享，需要大量同步(lock, semaphore, etc.)
MP，资源独享，进程间通信(收集与合并执行结果)

Notes:

- G.I.L.
- 序列化

# core problems

## SO, 关于fork()我们能够做哪些？

方案2：对象池模型

- 一种创建类的设计模式
- 适用场景：
  - 类实例初始化开销很大
  - 实例化比例高，同一时刻实例存在数量少/有限

# core problems

SO, 关于fork()我们能够做哪些？

方案3：也许我们可以对Windows期待更多?

# core problems

## 为什么需要源码?

编译时插桩，是为了

- 其一，fork server的关键部分(包括调用fork())
- 其二，获取目标程序的执行trace

如果我们不考虑fork()，那么我们只需关心如何得到trace.

# core problems

SO, 如何得到trace呢?

- 动态插桩
- 静态插桩
- 借助硬件
- 借助调试器

# core problems

SO, 如何得到trace呢?

- 动态插桩

  编写pintool

- 借助调试器

  IDA: patch 0xcc to all branches

  Debugger: monitor exceptions

  - 自己编写
  - windbg / pykd
  - TitanEngine

# other problems

## 如何决定何时终止某次执行?

经典的方法：timeout, 即set a timer

更好的方法：结合trace，单位时间内的计数增量比率小于某阈值

```
while Running:
    curr = count(trace)
    if curr > LLIMIT and ((curr - prev) << 10) < prev:
        kill process
    prev = curr
```

# other problems

目标程序非正常关闭的后遗症?

- 编写小程序，定期的查找目标窗口并关闭，例如FOE.
- 手工patch目标binary
- 使用debugger

# other problems

## 仍然不够快?

- 程序自身的优化 - pintool
- DLL重定位
- 将程序和样本全都放在ram disk上
- 样本裁剪
- 其他编程语言 C++, Go?
- 概率

# some bugs

## packages limit

multiprocessing

- locks
- process
- server, client
- rpc
- …

By the way …

why a distributed fuzzer?

# some bugs

## out of memory

```
Python 2.7.10 (default, May 23 2015, 09:40:32)
Type "help" for more information.
>>> a = 1
>>> b = 1
>>> c = 1000
>>> d = 1000
>>> id(x)?
```

# results

when we test on utilities of swftools ... 

what about the flashplayer ? 

and some other heavy targets ?

to be better, man

# special thanks

@nforest @wushi @promised_lu

# Many Thanks

Close

Replay