Revery—从漏洞PoC到可利用状态（迈向自动化攻击的一小步）

王琰　　中科院信息工程研究所

个人简介

姓名：王琰
专业：信息安全
2014级硕博连读
2019年7月博士毕业（导师: 邹维）

多次取得优异成绩

NeSE 战队队长

高难度漏洞利用题目

研究方向: 自动化漏洞利用

漏扫插件与IPS规则

恶意样本分析

重大安全事件响应

上海交通大学——信息安全工程学院

绿盟科技——核心技术部

中科院信息工程研究所——第六研究室

2007

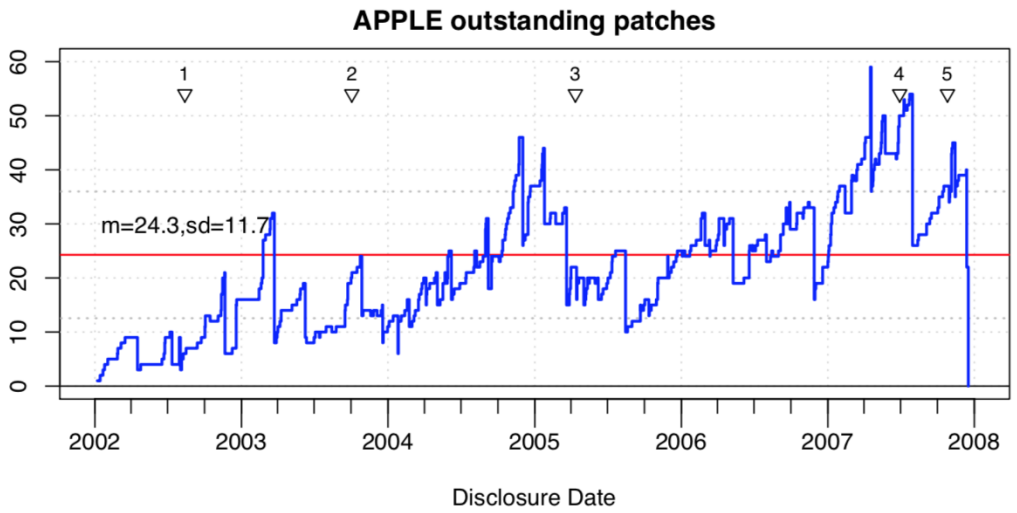
2011

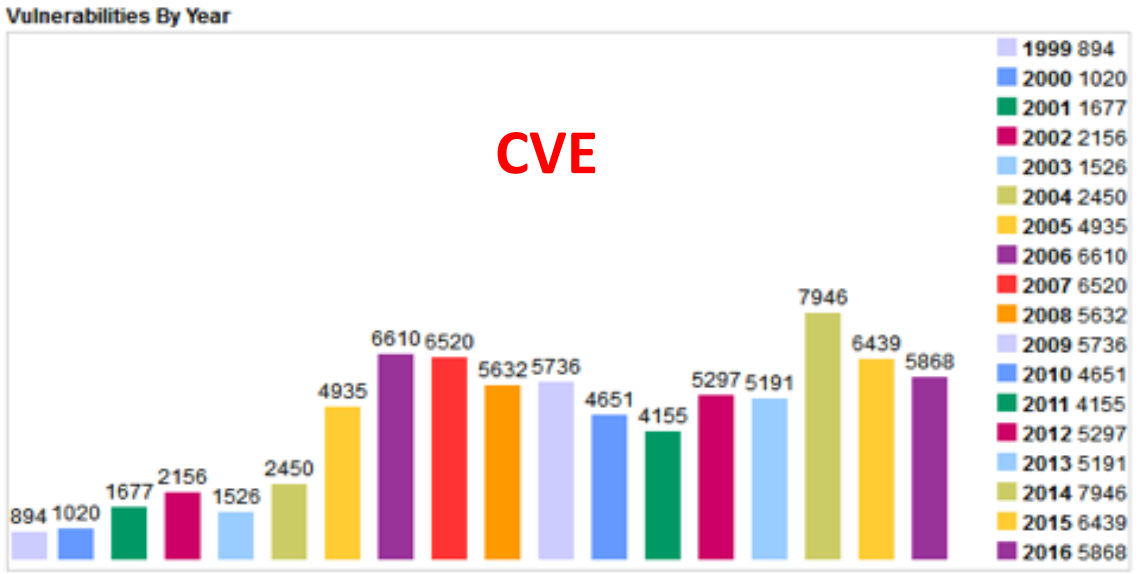
2014

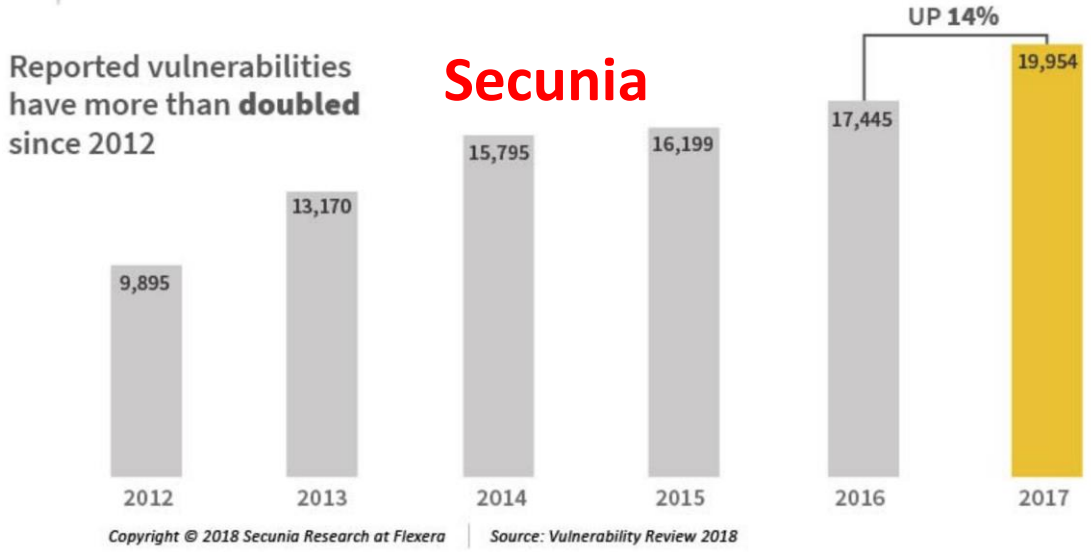
Motivation

自动化漏洞利用的必要性

- **漏洞修补**
  - 漏洞挖掘自动化程度高
  - 漏洞数量屡创新高
    - 2016年，NVD 收录6439条CVE漏洞信息；CNNVD收录8870条；CNVD收录10293条
  - 漏洞响应及修补费时费力
  - Google Project Zero 90天披露原则
  - 需求：自动评估漏洞，优先修补可利用的



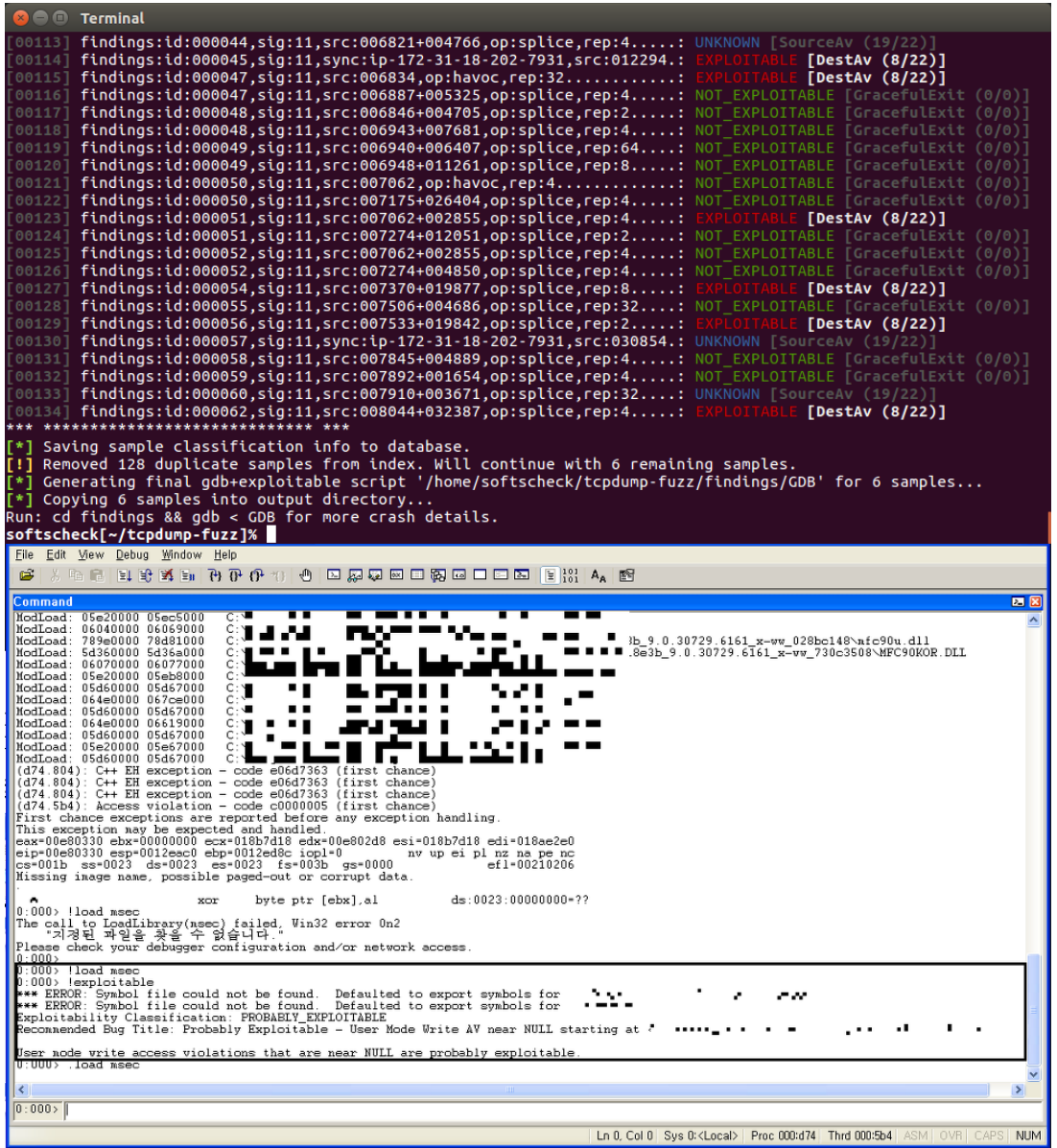




自动化漏洞利用的必要性

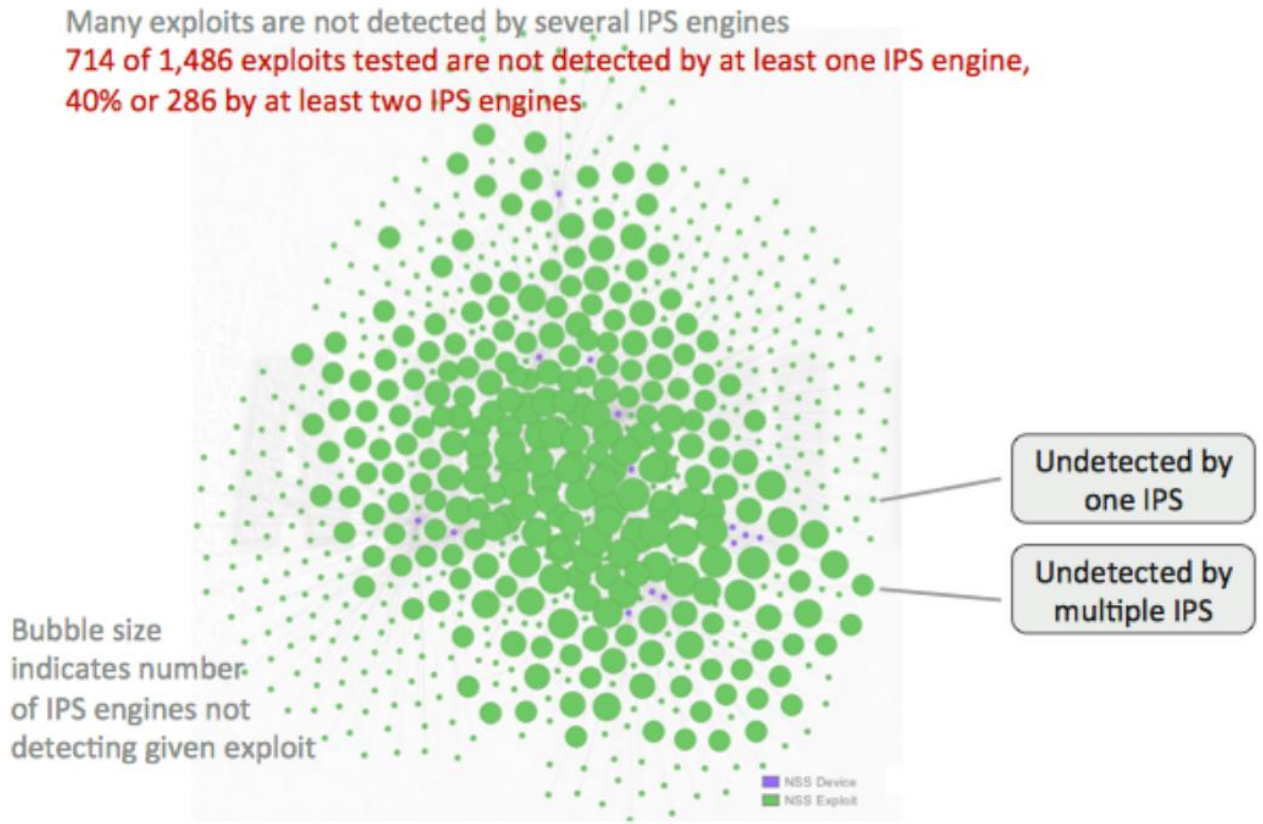
- 漏洞修补
  - 漏洞挖掘自动化程度高
  - 漏洞数量屡创新高
  - 漏洞响应及修补费时费力
  - Google Project Zero 90天披露原则
  - 需求：自动评估漏洞，优先修补可利用的

- **漏洞评估**
  - GDB exploitable插件
    - 基于漏洞类型判定
  - Windbg !exploitable插件
    - 基于崩溃点基本块模式判定
  - HCSIFTER方案
    - 修复堆metadata，及模式判定
  - 需求：自动生成exploit，评估漏洞

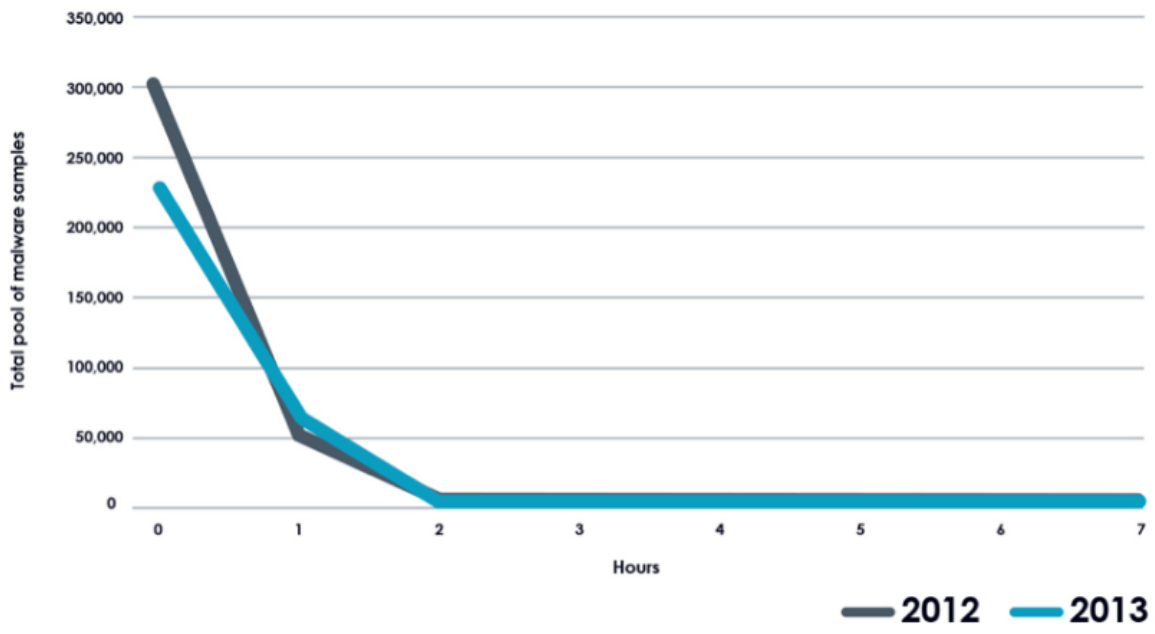


自动化漏洞利用的必要性

- 漏洞修补
  - 漏洞挖掘自动化程度高
  - 漏洞数量屡创新高
  - 漏洞响应及修补费时费力



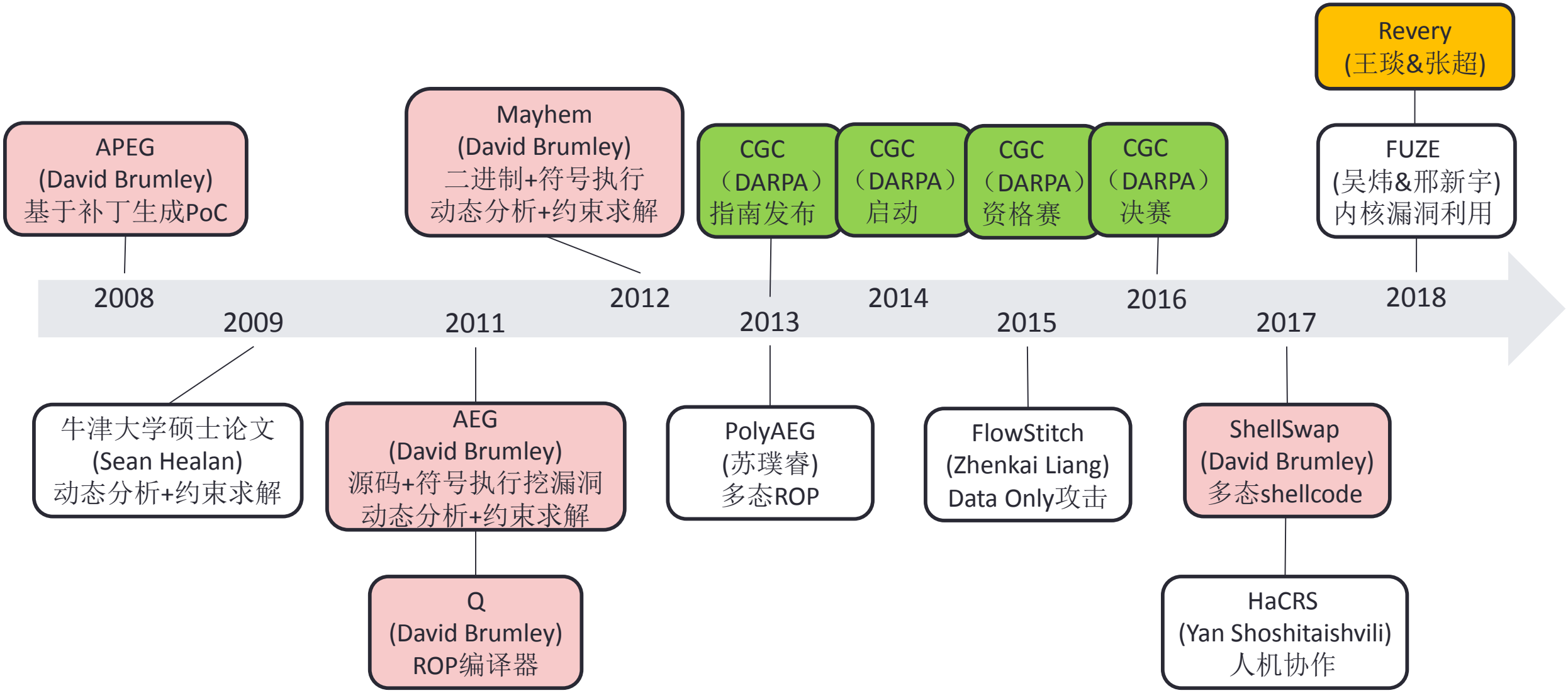
- **攻击检测**
  - 恶意软件生存周期短
    - FireEye数据：大部分少于2小时
  - 漏洞利用形态变化
    - NSS数据：IPS引擎漏检严重
  - 需求：自动分析漏洞、生成多态利用



自动化漏洞利用的必要性

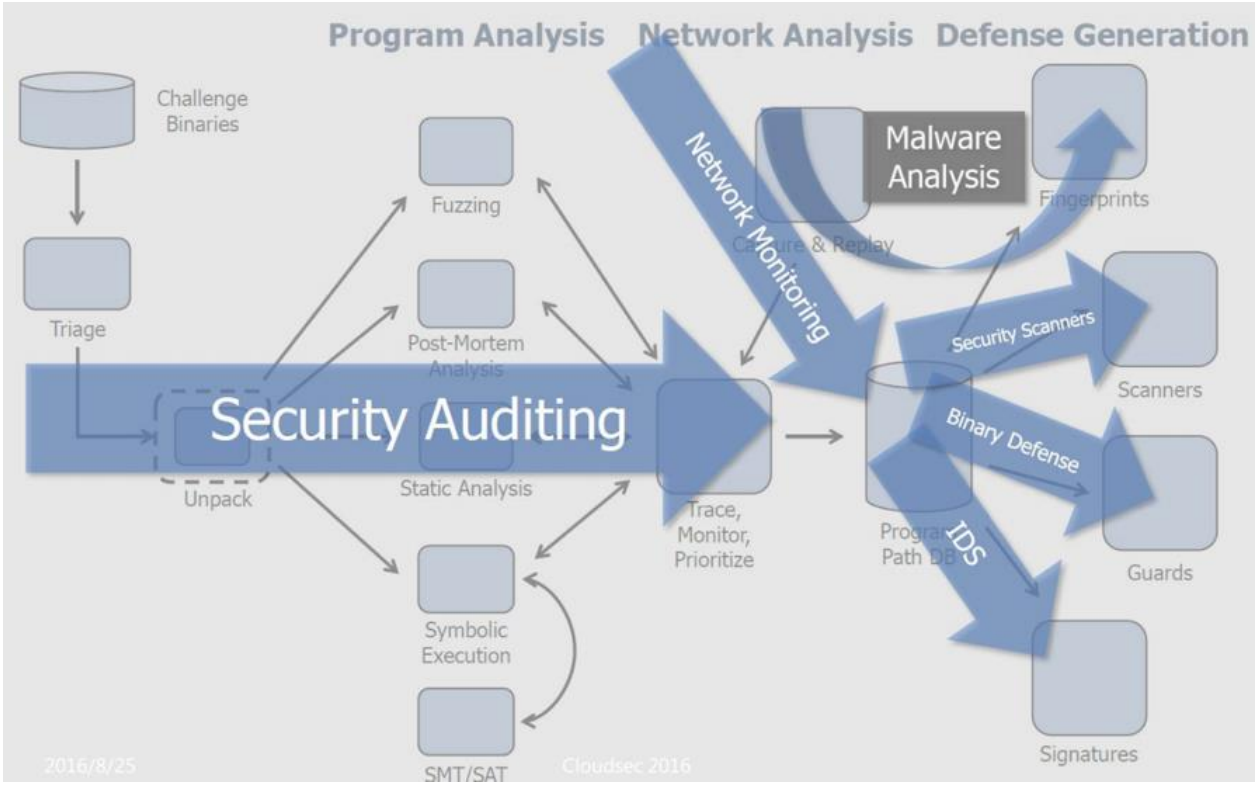
- **攻防实践**
  - 漏洞分析是个脑力活 + 体力活
    - DEFCON CTF 3天
  - 攻防的未来是机器对抗
    - The rising of machine. (DEFCON 2016)
  - 需求：自动化攻防

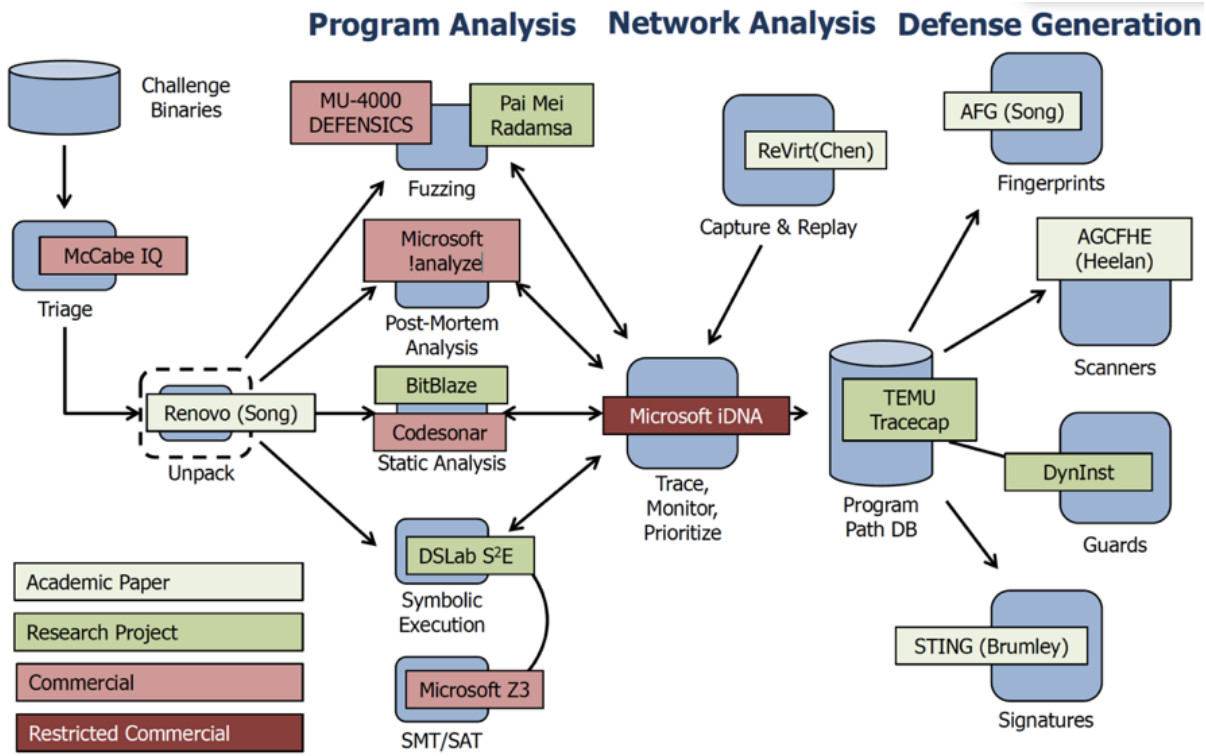
自动化漏洞利用的进展


APEG
(David Brumley)
基于补丁生成PoC
Mayhem
(David Brumley)
二进制+符号执行
动态分析+约束求解
CGC
（DARPA）
指南发布
CGC
（DARPA）
启动
CGC
（DARPA）
资格赛
CGC
（DARPA）
决赛
Revery
(王琰&张超)
FUZE
(吴炜&邢新宇)
内核漏洞利用
2008
2009
2011
2012
2013
2014
2015
2016
2017
2018
牛津大学硕士论文
(Sean Healan)
动态分析+约束求解
AEG
(David Brumley)
源码+符号执行挖漏洞
动态分析+约束求解
Q
(David Brumley)
ROP编译器
PolyAEG
(苏璞睿)
多态ROP
FlowStitch
(Zhenkai Liang)
Data Only攻击
ShellSwap
(David Brumley)
多态shellcode
HaCRS
(Yan Shoshitaishvili)
人机协作

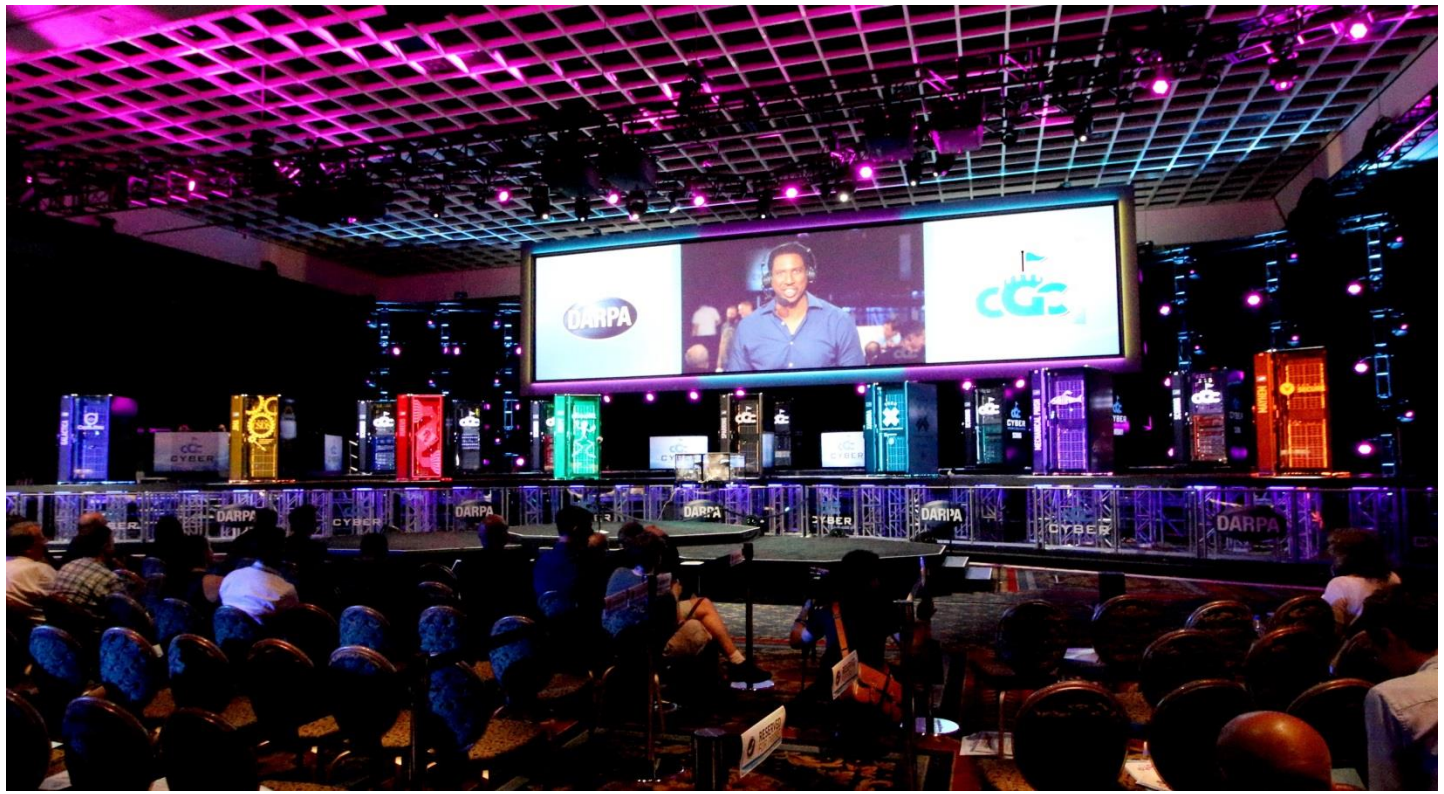
自动化漏洞利用的技术基础

- DARPA认为相关技术已经可贯通，自动化攻防在当前技术体系下已经可以实现。





业界的尝试与探索



- DARPA Cyber Grand Challenge (CGC)
  - 第一次全自动攻防，体现了 Grand Challenge风格
  - 裁剪的定制系统，突出了攻防核心任务
  - 冠军机器参与了DEFCON CTF 2016 人机大战
- 我们的战绩
  - CGC资格赛防御第一，决赛攻击第二。DEFCON CTF第二

- 2017 RHG
  - Linux环境

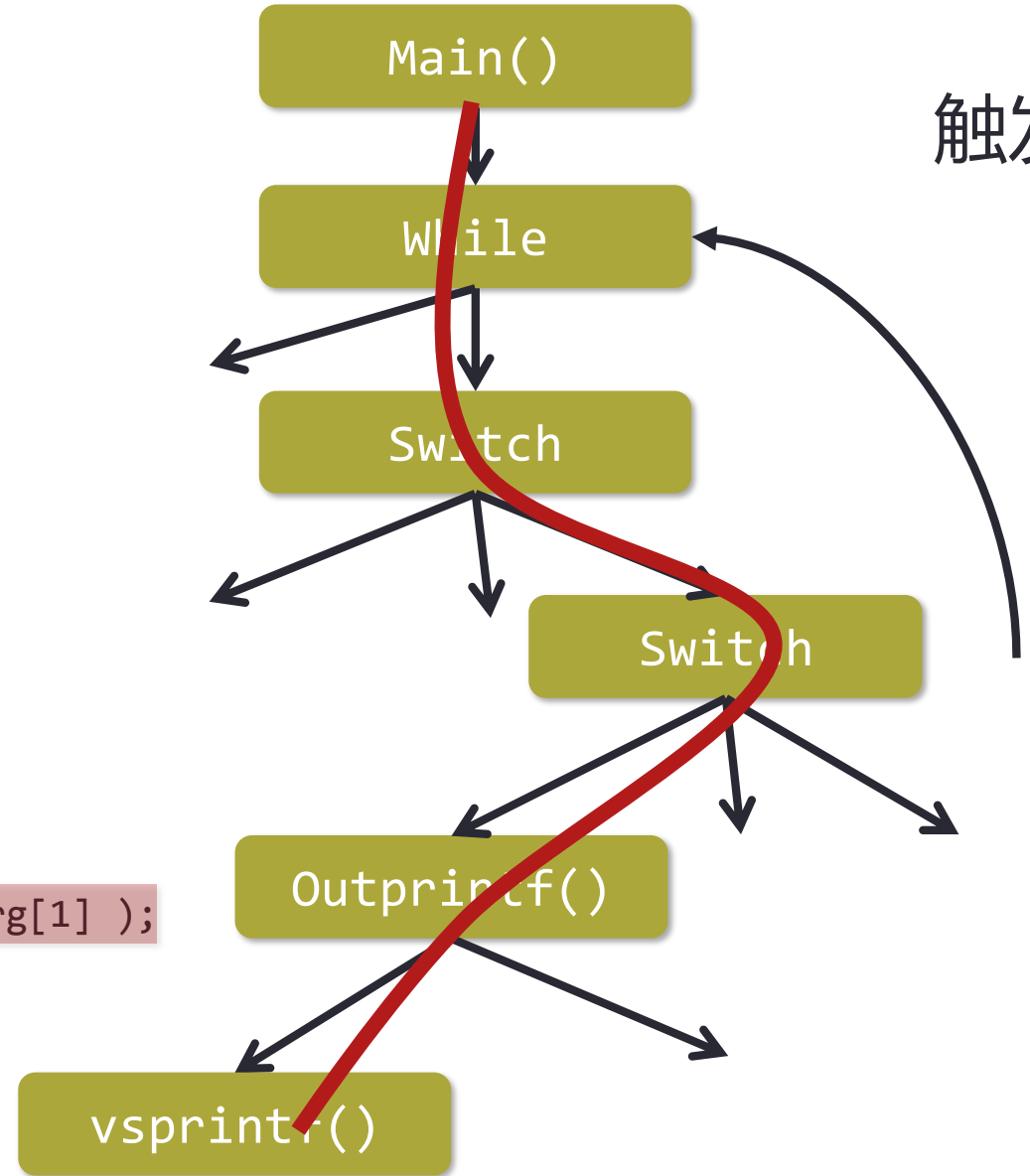
- 2018 DEFCON China
  - Linux环境，人机大战

What is AEG?

漏洞示例：CVE-2009-4270

```
int outprintf( const char *fmt, … )
{
  int count; char buf[1024]; va_list args;
  va_start( args, fmt );
  count = vsprintf( buf, fmt, args );
  outwrite( buf, count ); // print out
}

int main( int argc, char* argv[] )
{
  const char *arg;
  while( (arg = *argv++) != 0 ) {
    switch ( arg[0] ) {
    case '-': {
      switch ( arg[1] ) {
      case 0:
      …
      default:
        outprintf( "unknown switch %s\n", arg[1] );
      }
    }
    default: …
    }
  …
```



触发漏洞的要素：

- 路径约束
- 漏洞约束

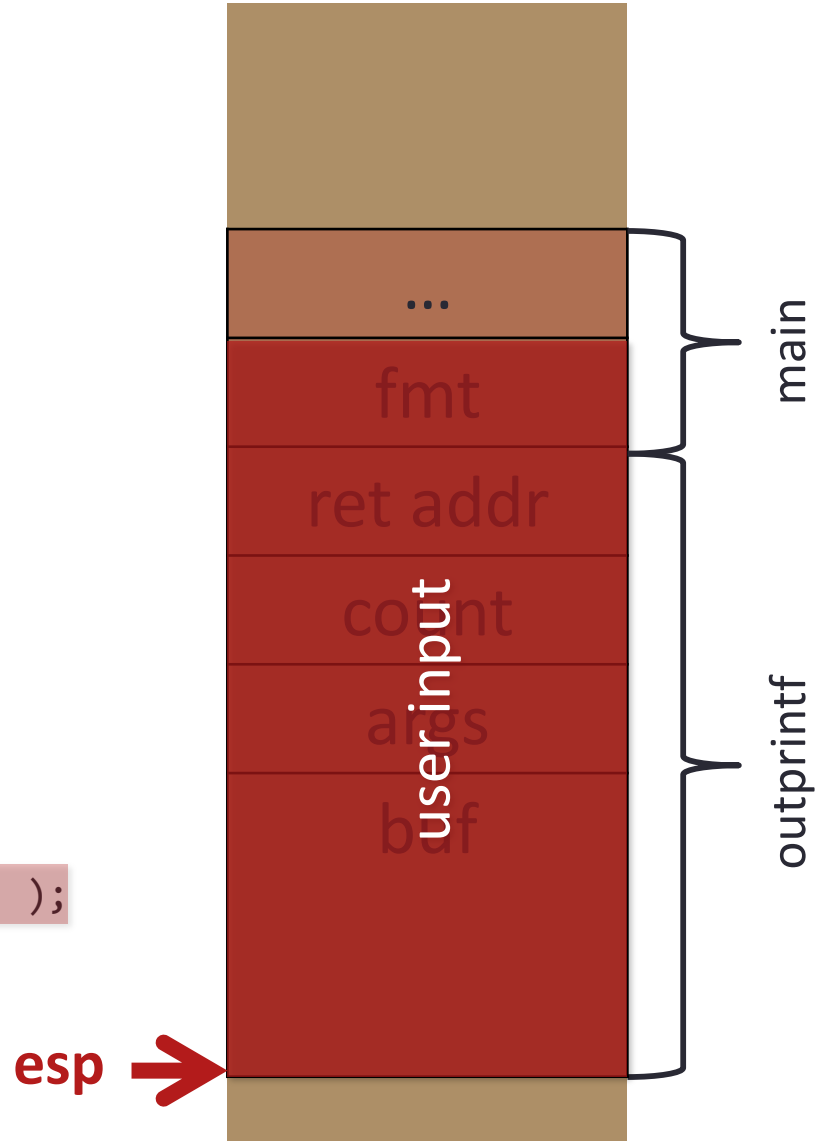
解决方案：

- 符号执行
- 模糊测试

漏洞利用

```
int outprintf( const char *fmt, … )
{
  int count; char buf[1024]; va_list args;
  va_start( args, fmt );
  count = vsprintf( buf, fmt, args );
  outwrite( buf, count ); // print out
}

int main( int argc, char* argv[] )
{
  const char *arg;
  while( (arg = *argv++) != 0 ) {
    switch ( arg[0] ) {
    case '-': {
      switch ( arg[1] ) {
      case 0:
      …
      default:
        outprintf( "unknown switch %s\n", arg[1] );
      }
    }
    default: …
    }
  …
```



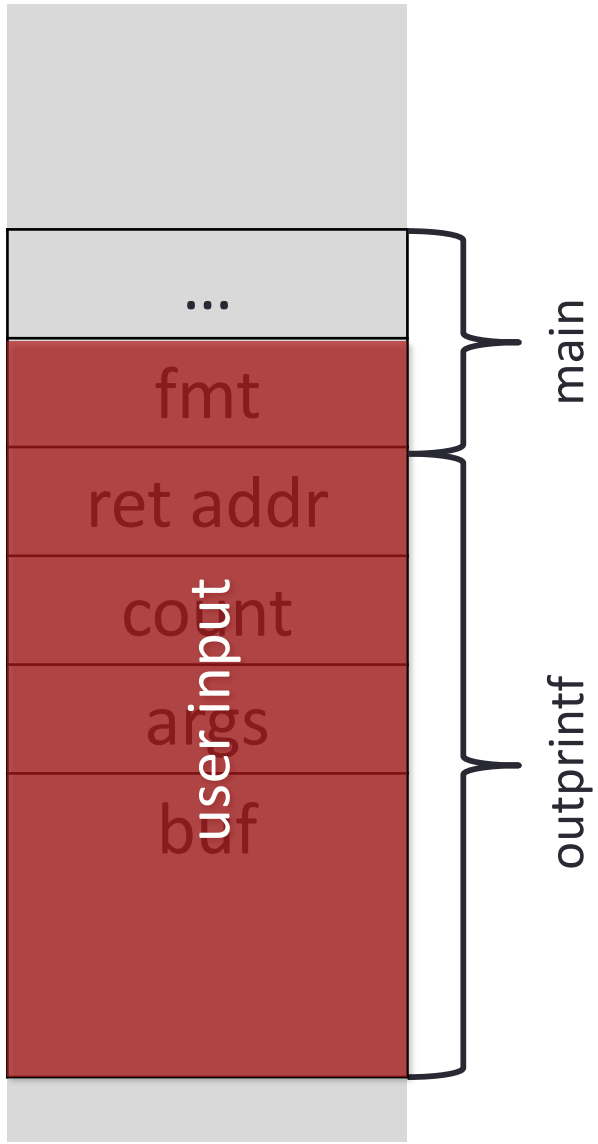
漏洞利用的要素：

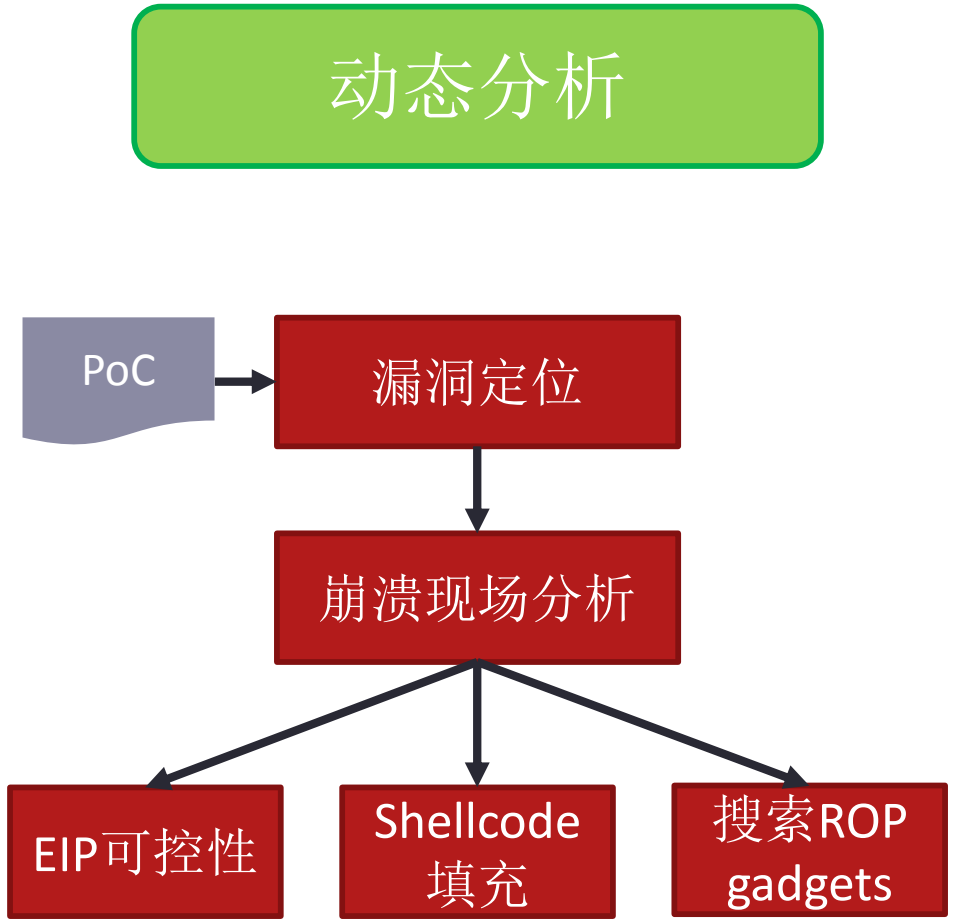
- 触发漏洞
  - 路径约束
  - 漏洞约束
- 利用漏洞
  - Shellcode约束
  - EIP约束
  - 内存布局约束
  - 防御绕过约束

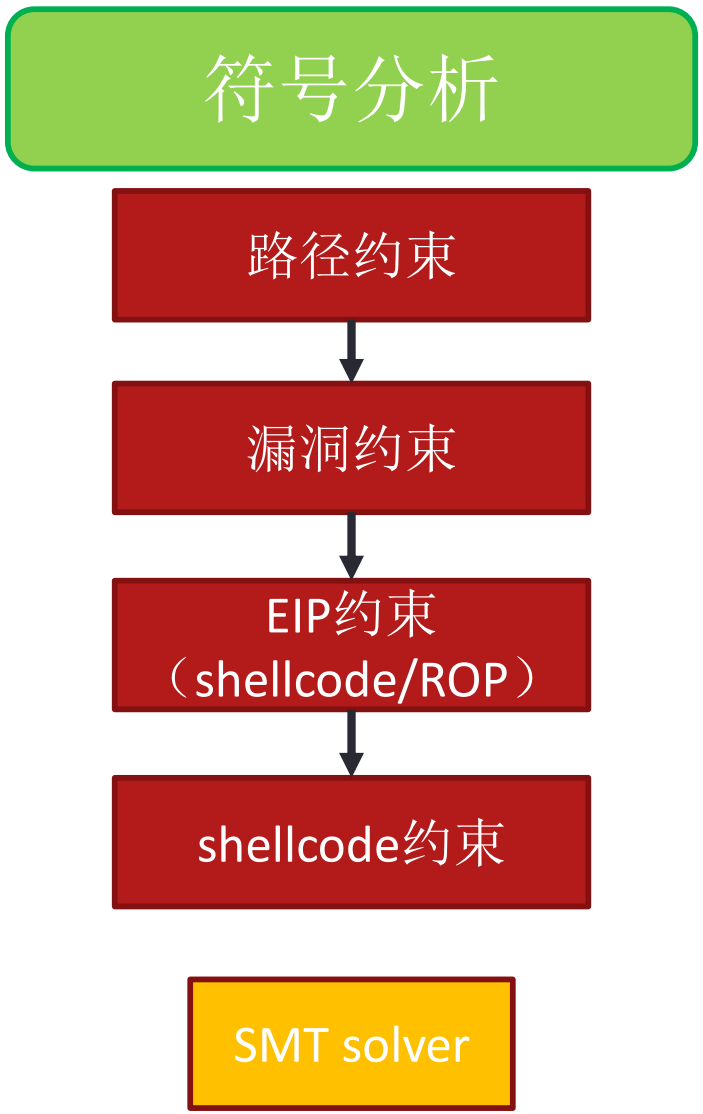
解决方案：

- 符号执行
- 模糊测试

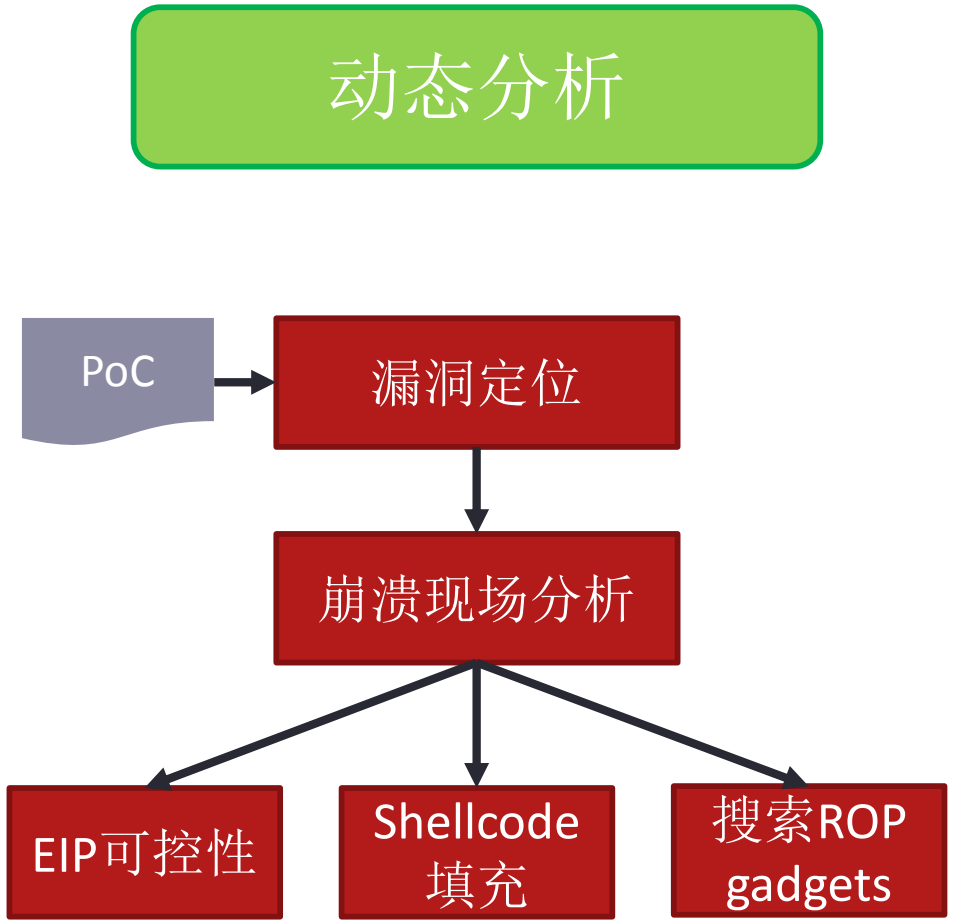
AEG (Automatic Exploit Generation)







AEG 挑战：PoC不可利用



- 适用于易于控制EIP的漏洞类型
  - 栈溢出
  - 格式化字符串
- 不适用于其他漏洞类型
  - 堆溢出...

PoC所在路径可能无法利用（即使人工参与）

Exploit derivability

PoC所在路径可能无法利用（即使人工参与）

寻找可利用的其他路径

Revery: From Proof-of-Concept to Exploitable

(One Step towards Automatic Exploit Generation)

Yan Wang*
wangyan9077@iie.ac.cn
Institute of Information Engineering,
Chinese Academy of Sciences
Beijing, China

Chao Zhang†
chaoz@tsinghua.edu.cn
Institute for Network Sciences and
Cyberspace, Tsinghua University
Beijing, China

Xiaobo Xiang*
xiangxiaobo@iie.ac.cn
Institute of Information Engineering,
Chinese Academy of Sciences
Beijing, China

Zixuan Zhao*
zhaozixuan@iie.ac.cn
Institute of Information Engineering,
Chinese Academy of Sciences
Beijing, China

Wenjie Li*
liwenjie@iie.ac.cn
Institute of Information Engineering,
Chinese Academy of Sciences
Beijing, China

Xiaorui Gong*†
gongxiaorui@iie.ac.cn
Institute of Information Engineering,
Chinese Academy of Sciences
Beijing, China

Bingchang Liu*
liubingchang@iie.ac.cn
Institute of Information Engineering,
Chinese Academy of Sciences
Beijing, China

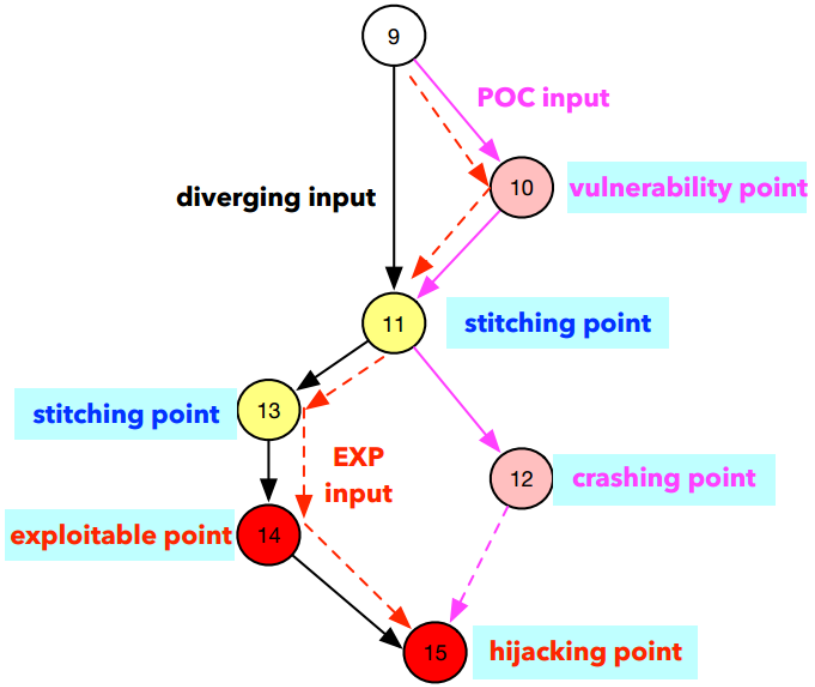
Kaixiang Chen
ckx1025ckx@gmail.com
Institute for Network Sciences and
Cyberspace, Tsinghua University
Beijing, China

Wei Zou*
zouwei@iie.ac.cn
Institute of Information Engineering,
Chinese Academy of Sciences
Beijing, China

To appear in ACM CCS 2018.

Example

```
1.  struct Type1 { char[8] data;                                    };
2.  struct Type2 { int status;    int* ptr;  void init(){…};        };
3.  int (*handler)(const int*) = …;
4.  struct{Type1* obj1; Type* obj2;} gvar = {};
5.  int foo(){
6.    gvar.obj1 = new Type1;
7.    gvar.obj2 = new Type2;
8.    gvar.obj2->init();  // resulting different statuses
9.    if(vul)
10.     scanf("%s", &gvar.obj1->data);     // vulnerability point
11.   if(gvar.obj2->status)                // stitching point
12.     res = *gvar.obj2->ptr;             // crashing point
13.   else                                 // stitching point
14.     *gvar.obj2->ptr = read_int();      // exploitable point
15.   handler(gvar.obj2->ptr);             // hijacking point
16.   return res;
17. }
```

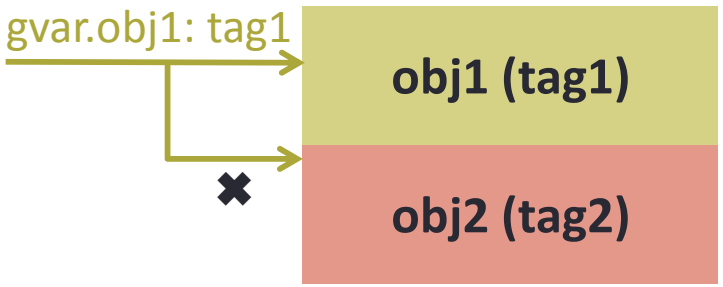


- 问题：PoC路径 (9->10->11->12->15) 无法利用
- 思路：回退PoC路径，找一条替代路径，进入可利用状态（EIP控制，或者任意写）。
- 回退到PoC路径什么位置？　怎么寻找替代路径？　怎么跳转到替代路径，进入可利用状态?

漏洞点　　模糊测试 (9->11->13->14)　　路径拼接 (11 is the stitching point)

漏洞分析：定位漏洞点

```
1.  struct Type1 { char[8] data;                                    };
2.  struct Type2 { int status;    int* ptr;  void init(){…};        };
3.  int (*handler)(const int*) = …;
4.  struct{Type1* obj1; Type* obj2;} gvar = {};
5.  int foo(){
6.    gvar.obj1 = new Type1;
7.    gvar.obj2 = new Type2;
8.    gvar.obj2->init();  // resulting different statuses
9.    if(vul)
10.     scanf("%s", &gvar.obj1->data);     // vulnerability point
11.   if(gvar.obj2->status)                // stitching point
12.     res = *gvar.obj2->ptr;             // crashing point
13.   else                                 // stitching point
14.     *gvar.obj2->ptr = read_int();      // exploitable point
15.   handler(gvar.obj2->ptr);             // hijacking point
16.   return res;
17. }
```



每个内存对象，关联一个污点标签tag（对象出生点），以及一个状态（未初始化、busy、已释放）

内存安全性访问原则：

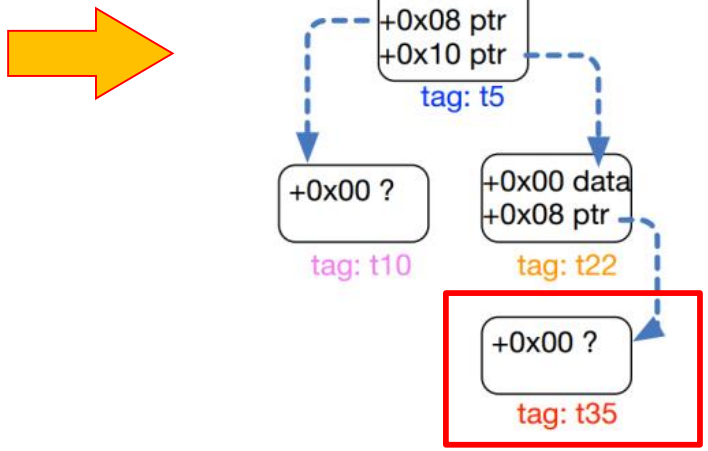
- **V1:** 访问预期的对象：
  - 指针tag_ptr与目标对象污点标签tag_obj相同
- **V2:** 读取busy状态的内存。
- **V3:** 写入非释放状态的内存。

漏洞点检测示例：

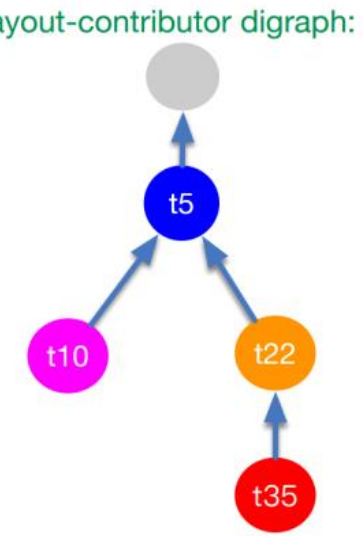
- 第10行为漏洞点，gvar.obj1指针含有污点标签tag1，当溢出发生时访问了obj2，而obj2的内存块标签为tag2，造成tag不匹配，违背了**V1**原则。

漏洞分析：识别异常对象布局

```
1.  struct Type1 { char[8] data;                                  };
2.  struct Type2 { int status;    int* ptr;  void init(){…};      };
3.  int (*handler)(const int*) = …;
4.  struct{Type1* obj1; Type* obj2;} gvar = {};
5.  int foo(){
6.    gvar.obj1 = new Type1;
7.    gvar.obj2 = new Type2;
8.    gvar.obj2->init();  // resulting different statuses
9.    if(vul)
10.     scanf("%s", &gvar.obj1->data);     // vulnerability point
11.   if(gvar.obj2->status)                // stitching point
12.     res = *gvar.obj2->ptr;             // crashing point
13.   else                                 // stitching point
14.     *gvar.obj2->ptr = read_int();      // exploitable point
15.   handler(gvar.obj2->ptr);             // hijacking point
16.   return res;
17. }
```



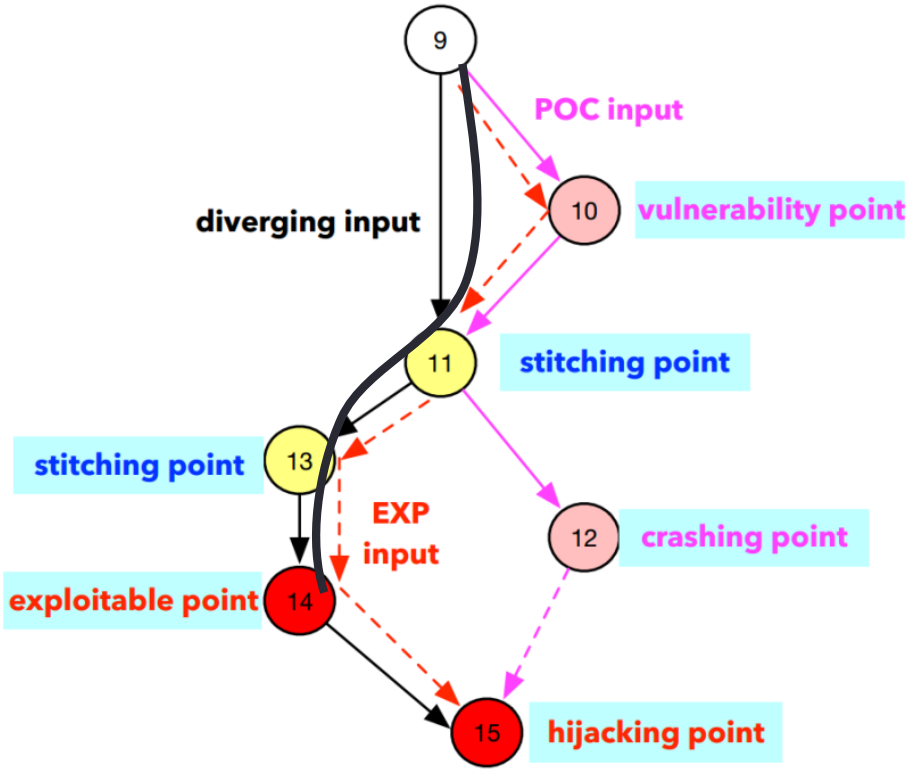
obj2



异常对象的内存布局分析：

- 异常对象：内容被非法篡改/控制的对象。
  - 例如，Obj1 溢出到obj2，则obj2为异常对象。
- 异常对象内存布局图：有向图，叶子节点为异常对象，根节点为全局变量，边为对象指向关系
- 内存布局贡献者图：基于异常对象内存布局图，添加指令标签
  - 节点的标签：创建对象的指令；边的标签：设置对象指向关系的指令。

探索替代路径（Diverging Path Exploration）



替代路径：9->11->13->14
替代路径通常不需要触发漏洞。

符号执行(Symbolic Execution)？

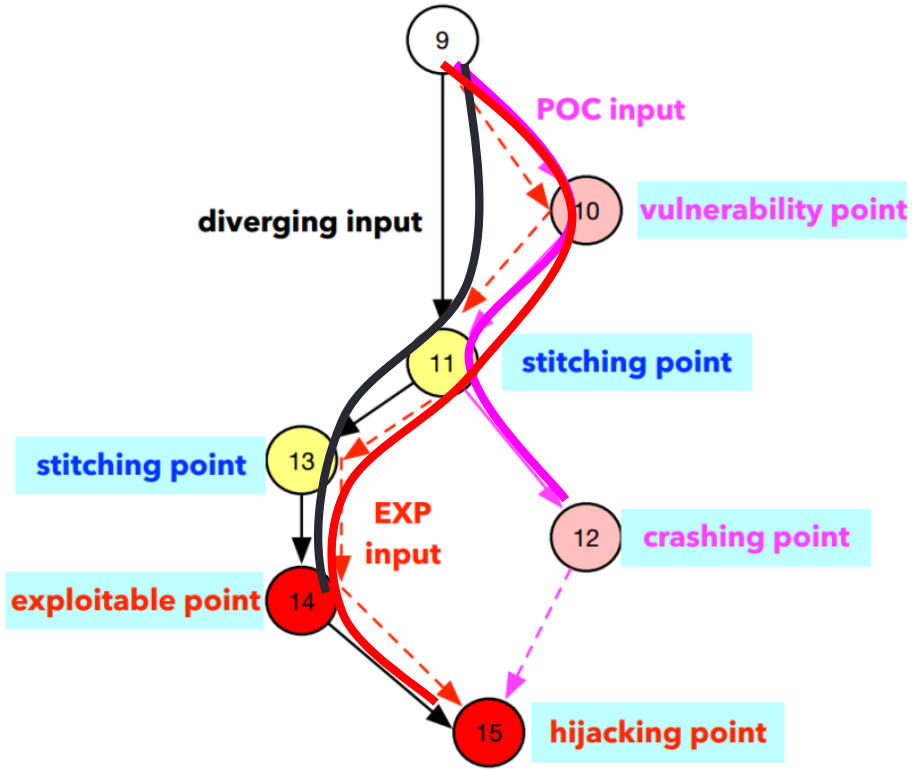
- 符号执行存在路径爆炸问题。
- 约束条件经常过于复杂而很难求解。
- 路径约束中缺少必要的漏洞利用数据约束：

——路径中malloc参数的符号化
——内存访问指令中的内存地址符号化

内存布局制导的定向Fuzzing

- 以异常对象的内存布局（layout-contributor digraph）为导向，采用定向Fuzzing探索替代路径。
- 记录指令命中次数，不断调整Fuzzing的方向（通过调整种子变异和种子优先级）
- 使用污点分析技术，在替代路径中寻找可利用状态（exploitable states）

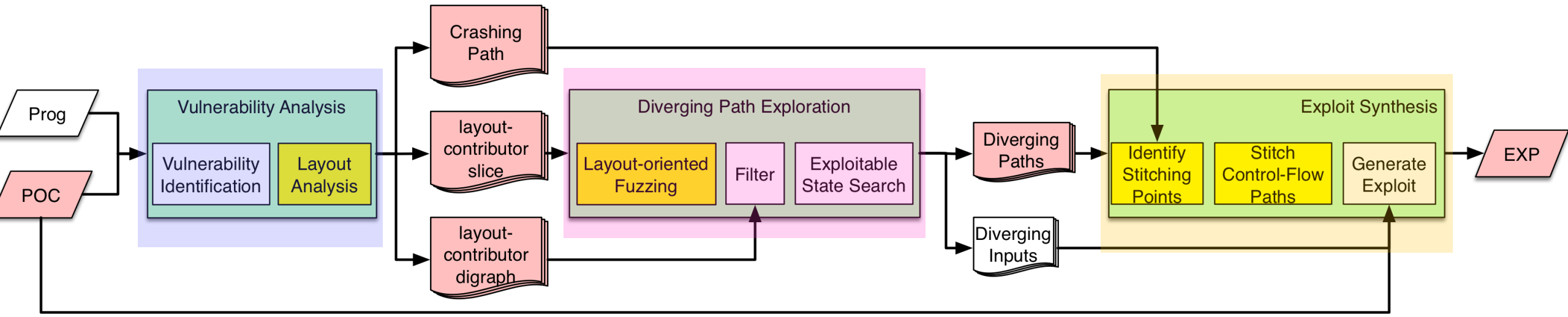
漏洞利用合成（Exploit Synthesis）



崩溃路径: 9->10->11->12
替代路径: 9->11->13->14
漏洞利用路径: 9->10->11->13->14->15

- 确定拼接点：保证漏洞触发并尽可能的重用替代路径。
- 路径拼接：重用替代路径中的子路径，组合生成漏洞利用路径。
  - 例如，重用替代路径中的**11->13->14** ，拼接到崩溃路径的**9->10->11**后面，生成利用路径**9->10->11->13->14->15**
- 利用生成：求解路径约束、漏洞利用数据约束等约束条件，生成漏洞利用。

Revery整体方案



- 分析漏洞位置和相关内存布局。
- 使用内存布局制导的定向Fuzzing探索替代路径，并且在替代路径中寻找可利用状态。
- 拼接控制流，并且求解相关约束生成漏洞利用。

实验结果（Evaluation）

	Name	CTF	Vul Type	Crash Type	Violation	Final State	EXP. Gen.	Rex	GDB Exploitable
	woO2	TU CTF 2016	UAF	heap error	V1	EIP hijack	YES	NO	Exploitable
CONTROL FLOW HIJACK	woO2_fixed	TU CTF 2016	UAF	heap error	V1	EIP hijack	YES	NO	Exploitable
	shop 2	ASIS Final 2015	UAF	mem read	V1	EIP hijack	YES	NO	UNKNOWN
	main	RHme3 CTF 2017	UAF	mem read	V1	mem write	YES	NO	UNKNOWN
	babyheap	SECUINSIDE 2017	UAF	mem read	V1	mem write	YES	NO	UNKNOWN
	b00ks	ASIS Quals 2016	Off-by-one	no crash	V1	mem write	YES	NO	Failed
	marimo	Codegate 2018	Heap overflow	no crash	V1	mem write	YES	NO	Failed
	ezhp	Plaid CTF 2014	Heap overflow	no crash	V1	mem write	YES	NO	Failed
	note1	ZCTF 2016	Heap Overflow	no crash	V1	mem write	YES	NO	Failed
	note2	ZCTF 2016	Heap Overflow	no crash	V1	unlink attack	NO	NO	Failed
EXPLOIT-ABLE STATE	note3	ZCTF 2016	Heap Overflow	no crash	V1	unlink attack	NO	NO	Failed
	fb	AliCTF 2016	Heap Overflow	no crash	V1	unlink attack	NO	NO	Failed
	stkof	HITCON 2014	Heap Overflow	no crash	V1	unlink attack	NO	NO	Failed
	simple note	Tokyo Westerns 2017	Off-by-one	no crash	V1	unlink attack	NO	NO	Failed
	childheap	SECUINSIDE 2017	Double Free	heap error	V1	-	NO	NO	Exploitable
	CarMarket	ASIS Finals 2016	Off-by-one	no crash	V1	-	NO	NO	Failed
FAILED	SimpleMemoPad	CODEBLUE 2017	Heap Overflow	no crash	-	-	NO	NO	Failed
	LFA	34c3 2017	Heap Overflow	no crash	-	-	NO	NO	Failed
	Recurse	33c3 2016	UAF	no crash	-	-	NO	NO	Failed

- 测试对象为来自**15**个CTF比赛的**19**个存在漏洞的程序。
- 其中**9**个程序可以自动化生成漏洞利用，**5**个可以自动化转化为可利用状态，**5**个无法自动利用。
- 可以自动化利用**内存读错误**，**堆错误**，甚至**不触发崩溃**的PoC。

Revery的局限：

- Revery以angr为平台进行开发，angr本身缺少必要的系统调用支持，无法运行真实程序。
- 目前无法自动化绕过ASLR，无法自动进行堆的原子化操作，无法自动化构造内存布局。

Takeaway
(总结)

AEG vs. Revery

- 传统AEG方案：
  - 依赖于PoC崩溃现场状态
  - 采用动态分析 + 符号执行方案
- 面临的挑战
  - PoC 崩溃现场可能无法利用
  - 符号执行 可扩展性差
  - 不支持堆漏洞利用

- Revery解决方案
  - 探索PoC之外的其他程序路径，寻找可利用的程序状态
  - 采用模糊测试（而不是符号执行）来探索exploitable程序路径
  - 利用关键内存操作指令制导模糊测试，快速筛选有价值的exploitable程序路径
  - 利用符号执行拼接PoC路径和exploitable路径，同时触发漏洞以及可利用的程序状态

Revery 仅仅将 AEG 向前推进了一小步

Roadblocks of AEG

- 漏洞利用的定义 (AH)
  - 定义异常程序状态（bof等）
  - 定义/隔离代码权限
- 漏洞利用样本生成 (BCDE)
  - 反推程序状态的前置/后置条件
  - 反推循环结束状态的前置条件
  - 反推到达特定状态的输入条件
- 漏洞利用原语组合F
  - 多漏洞/交互组合达到目标状态
- 环境操纵 (IJK)
  - 内存/堆风水
  - 时间分析，推断值、大小、位置等信息
  - 信息泄露，如侧信道等
- 状态空间表示 (G)
  - 多线程漏洞利用

The Automated Exploitation Grand Challenge

Tales of Weird Machines

Julien Vanegue

julien.vanegue@gmail.com

H2HC conference, Sao Paulo, Brazil

October 2013

- Infrastructure
  - 符号执行、污点分析、反汇编。。。

How to Improve AEG?

- AEG 研究
  - 从实践中发现问题
  - 从实践中寻找经验
  - 融合多种技术
  - 踏踏实实写代码
- AEG 比赛
  - 避免过度追求 0→1
  - 鼓励分阶段/分问题挑战
  - 细化比赛规则设计，鼓励创新

谢谢！

Q&A