Revery: from POC to EXP

Chao Zhang
Tsinghua University

About Me

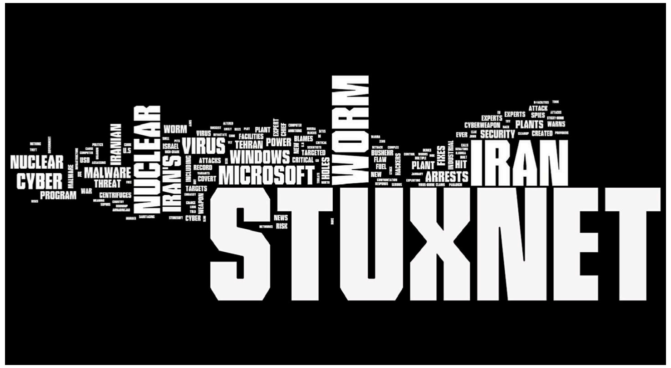
2004-2008-2013	→	2013-2016	→	2016-present
北京大學 PEKING UNIVERSITY		Berkeley UNIVERSITY OF CALIFORNIA		清華大學 Tsinghua University

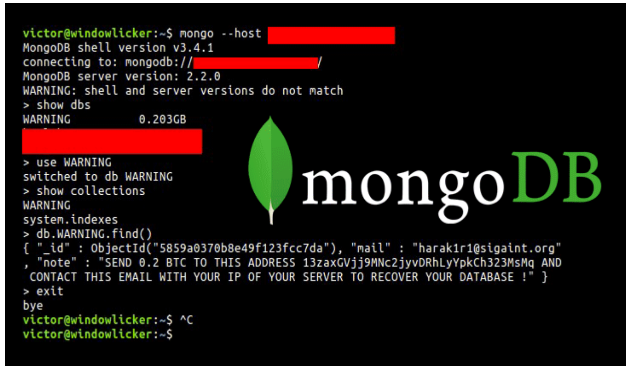
▪ **Hack for fun**	*software and system security*
• Automated vul. discovery:	**CSS TSec** 2nd Place (300+ vulnerabilities, 200+ CVE)
• Automated exploit generation:	**CSS TSec** Breakthrough Prize (1st place)
• Automated exploit mitigation:	**Microsoft BlueHat** Prize (Special Recognition Award)
• Automated attack & defense:	**DARPA CGC** (1st in defense 2015, 2nd in offense 2016)
• Manual hacking:	**DEFCON CTF** (2nd in 2016, 5th in 2015 and 2017)

▪ **Awards/Honors**	
• Tsinghua University Rising Star	2019
• MIT TR35 China	2018
• Qiu Shi Outstanding Young Scholar	2018
• Thousand Youth Talents Plan	2018
• Young Elite Scientists Sponsorship (CAST)	2017

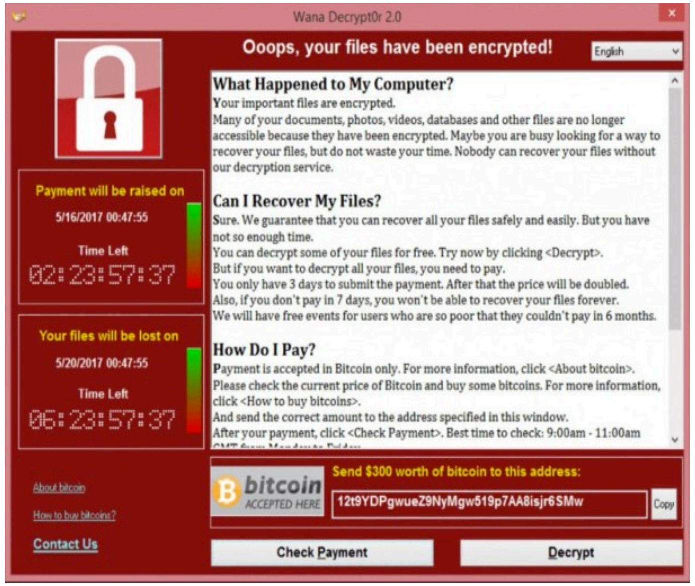
Vulnerability: Ghost in Cyberspace

- Valuable assets, root causes of most security incidents





```
victor@windowlicker:~$ mongo --host [redacted]
MongoDB shell version v3.4.1
connecting to: mongodb://[redacted]/
MongoDB server version: 2.2.0
WARNING: shell and server versions do not match
> show dbs
WARNING          0.203GB
[redacted]
> use WARNING
switched to db WARNING
> show collections
WARNING
system.indexes
> db.WARNING.find()
{ "_id" : ObjectId("5859a0370b8e49f123fcc7da"), "mail" : "harak1r1@sigaint.org"
, "note" : "SEND 0.2 BTC TO THIS ADDRESS 13zaxGVjj9MNc2jyvDRhLyYpkCh323MsMq AND
 CONTACT THIS EMAIL WITH YOUR IP OF YOUR SERVER TO RECOVER YOUR DATABASE !" }
> exit
bye
victor@windowlicker:~$ ^C
victor@windowlicker:~$
```



Ooops, your files have been encrypted!

Payment will be raised on

5/16/2017 00:47:55

Time Left

02:23:57:37

Your files will be lost on

5/20/2017 00:47:55

Time Left

06:23:57:37

What Happened to My Computer?

Your important files are encrypted.

Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?

Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.

You can decrypt some of your files for free. Try now by clicking <Decrypt>.

But if you want to decrypt all your files, you need to pay.

You only have 3 days to submit the payment. After that the price will be doubled.

Also, if you don't pay in 7 days, you won't be able to recover your files forever.

We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?

Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.

Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.

And send the correct amount to the address specified in this window.

After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am

Send $300 worth of bitcoin to this address:

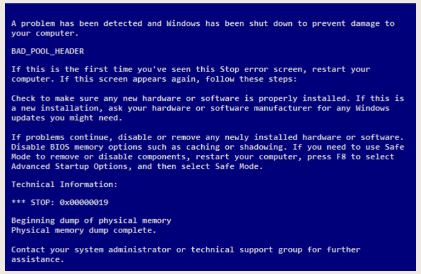
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

OPERATION AURORA
How To Respond To The Recent Microsoft Internet Explorer Vulnerability

Exploiting Vulnerabilities

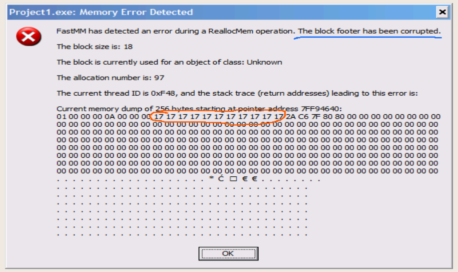
Victim Application

Trigger Vulnerabilities
(buffer overflow …)



```
A problem has been detected and Windows has been shut down to prevent damage to
your computer.

BAD_POOL_HEADER

If this is the first time you've seen this Stop error screen, restart your
computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is
a new installation, ask your hardware or software manufacturer for any Windows
updates you might need.

If problems continue, disable or remove any newly installed hardware or software.
Disable BIOS memory options such as caching or shadowing. If you need to use Safe
Mode to remove or disable components, restart your computer, press F8 to select
Advanced Startup Options, and then select Safe Mode.

Technical Information:

*** STOP: 0x00000019

Beginning dump of physical memory
Physical memory dump complete.

Contact your system administrator or technical support group for further
assistance.
```

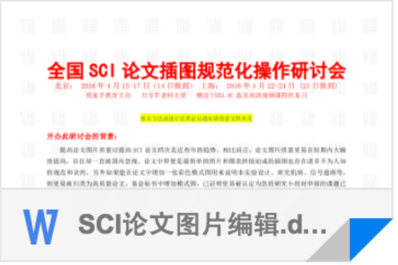
Tamper Program States
(Code、Data)



```
FastMM has detected an error during a ReallocMem operation. The block footer has been corrupted.

The block size is: 18

The block is currently used for an object of class: Unknown

The allocation number is: 97

The current thread ID is 0xF48, and the stack trace (return addresses) leading to this error is:

Current memory dump of 256 bytes starting at pointer address 7FF94640:
01 00 00 00 0A 00 00 00 17 17 17 17 17 17 17 17 17 17 17 2A C6 7F 80 80 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
. . . . . . . . . . . . . . . . . . . * Ć □ € € . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Bypass Existing Defenses
(DEP, ASLR, sandbox …)

Attack Vector
(via spam, website …)



全国 SCI 论文插图规范化操作研讨会

北京：2016年4月15-17日（14日报到） 上海：2016年4月22-24日（21日报到）

[illegible]

开办此研讨会的背景：

[illegible]

Break Tgt System
(malware, leak, control…)

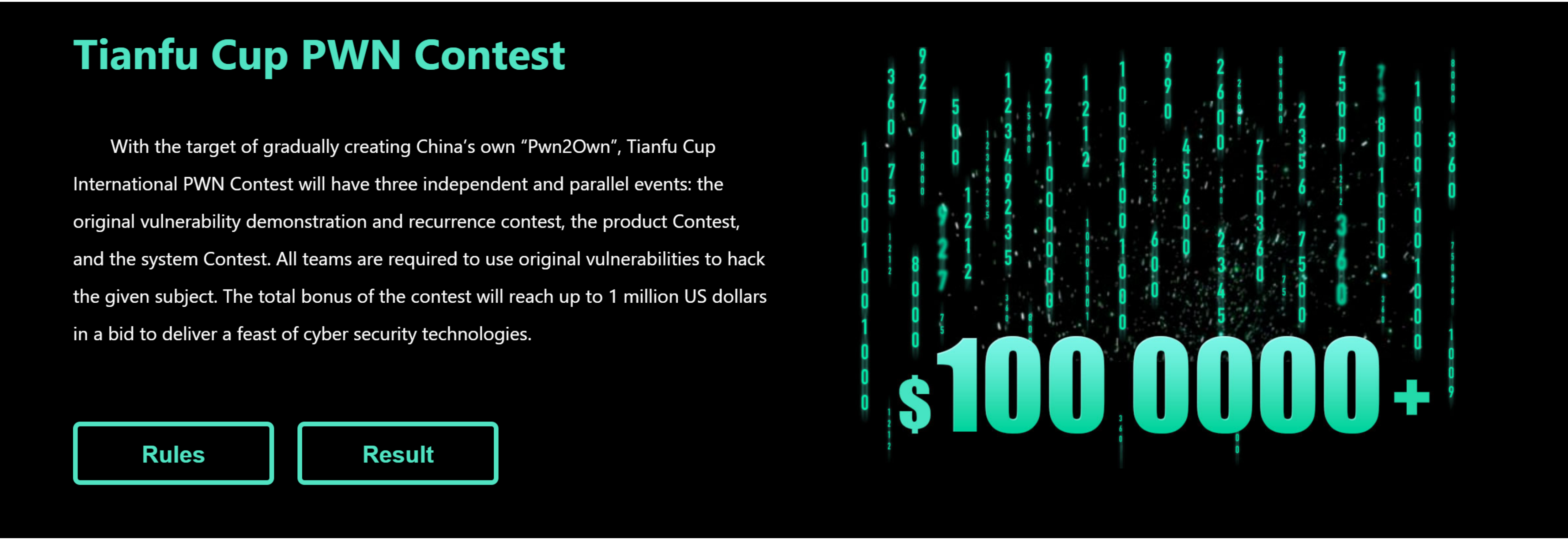
Exploiting in Practice

DEFCON CTF

(blue-lotus, Tea-Deliverer)

Pwn2Own

Exploiting in Practice



Tianfu Cup PWN Contest

With the target of gradually creating China’s own “Pwn2Own”, Tianfu Cup International PWN Contest will have three independent and parallel events: the original vulnerability demonstration and recurrence contest, the product Contest, and the system Contest. All teams are required to use original vulnerabilities to hack the given subject. The total bonus of the contest will reach up to 1 million US dollars in a bid to deliver a feast of cyber security technologies.

Only *patient* *experienced* hackers can do it.

Can machines **exploit vulnerabilities** like human, and even better than human?

Automated Exploit Generation (AEG)

Motivation

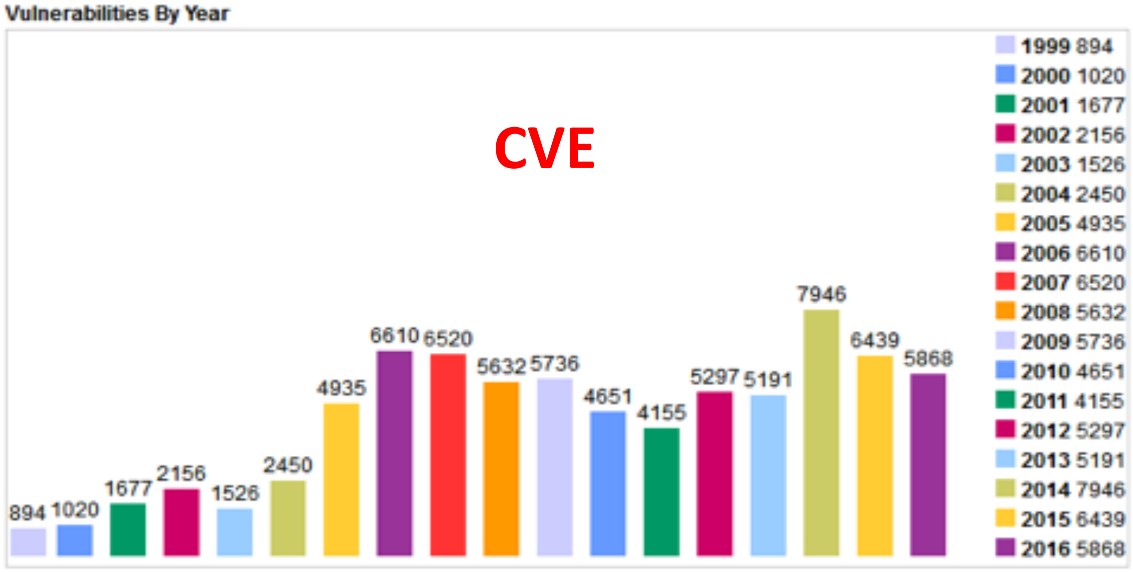
To better defend yourself,

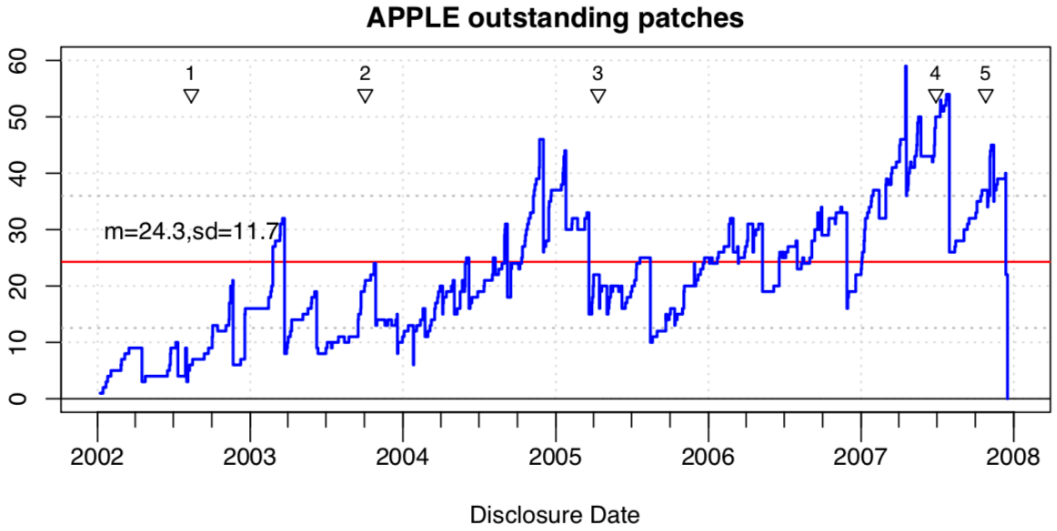
Know your enemy first.

Sun Tzu

Why AEG?

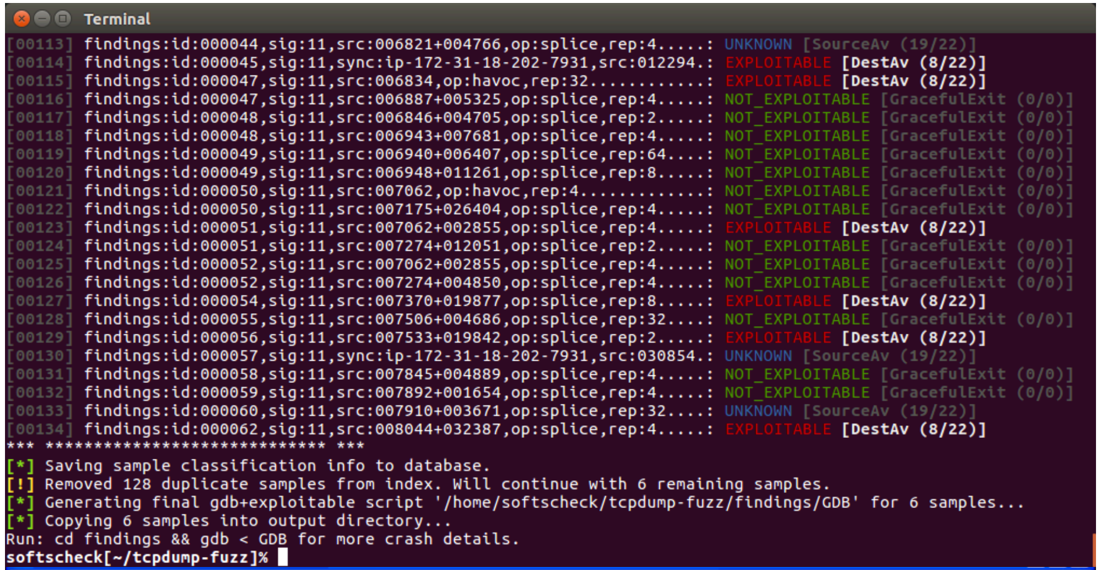
- **Fixing vulnerabilities**
  - Automated vul discovery solutions
  - High volume of vulnerabilities
  - Long time to fix one vulnerability
  - 90 days deadline (Google Project Zero)
  - Need: automated vul assessment
    - to prioritize vulnerabilities to fix
  - Case: Facebook 50M user info leaked.



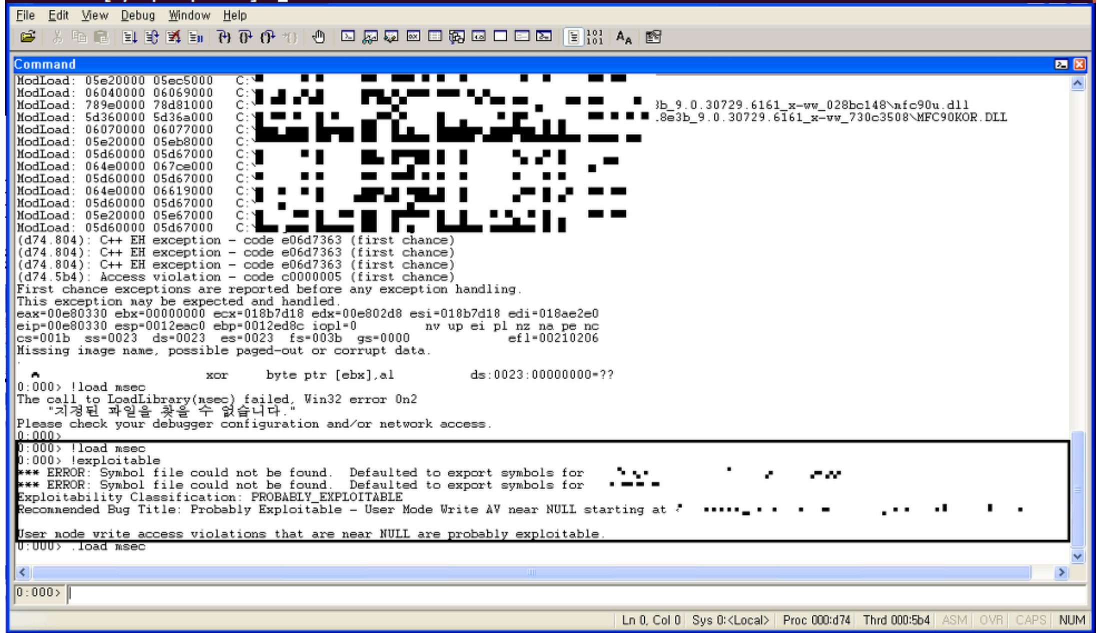


Why AEG?

- Fixing vulnerabilities
  - Need: automated vul assessment
- **Vulnerability Assessment**
  - GDB 'exploitable' plugin
    - Depends on vul type
  - WinDBG '!exploitable' plugin
    - Depends on basic block type
  - HCSIFTER (ASE' 17)
    - Recover heap metadata , vul pattern
  - Need: assess vulnerability with AEG



```
[00113] findings:id:000044,sig:11,src:006821+004766,op:splice,rep:4.....: UNKNOWN [SourceAv (19/22)]
[00114] findings:id:000045,sig:11,sync:ip-172-31-18-202-7931,src:012294.: EXPLOITABLE [DestAv (8/22)]
[00115] findings:id:000047,sig:11,src:006834,op:havoc,rep:32...........: EXPLOITABLE [DestAv (8/22)]
[00116] findings:id:000047,sig:11,src:006887+005325,op:splice,rep:4.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00117] findings:id:000048,sig:11,src:006846+004705,op:splice,rep:2.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00118] findings:id:000048,sig:11,src:006943+007681,op:splice,rep:4.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00119] findings:id:000049,sig:11,src:006940+006407,op:splice,rep:64....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00120] findings:id:000049,sig:11,src:006948+011261,op:splice,rep:8.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00121] findings:id:000050,sig:11,src:007062,op:havoc,rep:4............: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00122] findings:id:000050,sig:11,src:007175+026404,op:splice,rep:4.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00123] findings:id:000051,sig:11,src:007062+002855,op:splice,rep:4.....: EXPLOITABLE [DestAv (8/22)]
[00124] findings:id:000051,sig:11,src:007274+012051,op:splice,rep:2.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00125] findings:id:000052,sig:11,src:007062+002855,op:splice,rep:4.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00126] findings:id:000052,sig:11,src:007274+004850,op:splice,rep:4.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00127] findings:id:000054,sig:11,src:007370+019877,op:splice,rep:8.....: EXPLOITABLE [DestAv (8/22)]
[00128] findings:id:000055,sig:11,src:007506+004686,op:splice,rep:32....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00129] findings:id:000056,sig:11,src:007533+019842,op:splice,rep:2.....: EXPLOITABLE [DestAv (8/22)]
[00130] findings:id:000057,sig:11,sync:ip-172-31-18-202-7931,src:030854.: UNKNOWN [SourceAv (19/22)]
[00131] findings:id:000058,sig:11,src:007845+004889,op:splice,rep:4.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00132] findings:id:000059,sig:11,src:007892+001654,op:splice,rep:4.....: NOT_EXPLOITABLE [GracefulExit (0/0)]
[00133] findings:id:000060,sig:11,src:007910+003671,op:splice,rep:32....: UNKNOWN [SourceAv (19/22)]
[00134] findings:id:000062,sig:11,src:008044+032387,op:splice,rep:4.....: EXPLOITABLE [DestAv (8/22)]
*** ****************************** ***
[*] Saving sample classification info to database.
[!] Removed 128 duplicate samples from index. Will continue with 6 remaining samples.
[*] Generating final gdb+exploitable script '/home/softscheck/tcpdump-fuzz/findings/GDB' for 6 samples...
[*] Copying 6 samples into output directory...
Run: cd findings && gdb < GDB for more crash details.
softscheck[~/tcpdump-fuzz]%
```

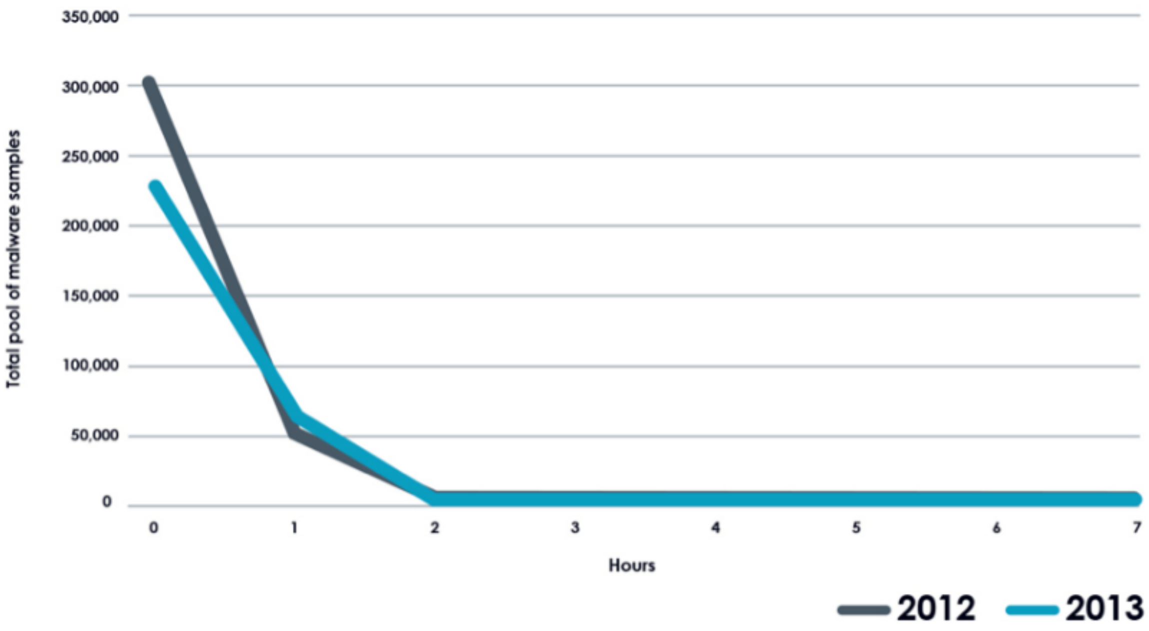


```
ModLoad: 05e20000 05ec5000   C:[redacted]
ModLoad: 06040000 06069000   C:[redacted]
ModLoad: 789e0000 78d81000   C:[redacted]3b_9.0.30729.6161_x-ww_028bc148\mfc90u.dll
ModLoad: 5d360000 5d36a000   C:[redacted].8e3b_9.0.30729.6161_x-ww_730c3508\MFC90KOR.DLL
ModLoad: 06070000 06077000   C:[redacted]
ModLoad: 05e20000 05eb8000   C:[redacted]
ModLoad: 05d60000 05d67000   C:[redacted]
ModLoad: 064e0000 067ce000   C:[redacted]
ModLoad: 05d60000 05d67000   C:[redacted]
ModLoad: 064e0000 06619000   C:[redacted]
ModLoad: 05d60000 05d67000   C:[redacted]
ModLoad: 05e20000 05e67000   C:[redacted]
ModLoad: 05d60000 05d67000   C:[redacted]
(d74.804): C++ EH exception - code e06d7363 (first chance)
(d74.804): C++ EH exception - code e06d7363 (first chance)
(d74.804): C++ EH exception - code e06d7363 (first chance)
(d74.5b4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00e80330 ebx=00000000 ecx=018b7d18 edx=00e802d8 esi=018b7d18 edi=018ae2e0
eip=00e80330 esp=0012eac0 ebp=0012ed8c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210206
Missing image name, possible paged-out or corrupt data.

                    xor     byte ptr [ebx],al          ds:0023:00000000=??
0:000> !load msec
The call to LoadLibrary(msec) failed, Win32 error 0n2
    "지정된 파일을 찾을 수 없습니다."
Please check your debugger configuration and/or network access.
0:000>
0:000> !load msec
0:000> !exploitable
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for [redacted]
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for [redacted]
Exploitability Classification: PROBABLY_EXPLOITABLE
Recommended Bug Title: Probably Exploitable - User Mode Write AV near NULL starting at [redacted]

User mode write access violations that are near NULL are probably exploitable.
0:000> .load msec
```

```
0:000>
```

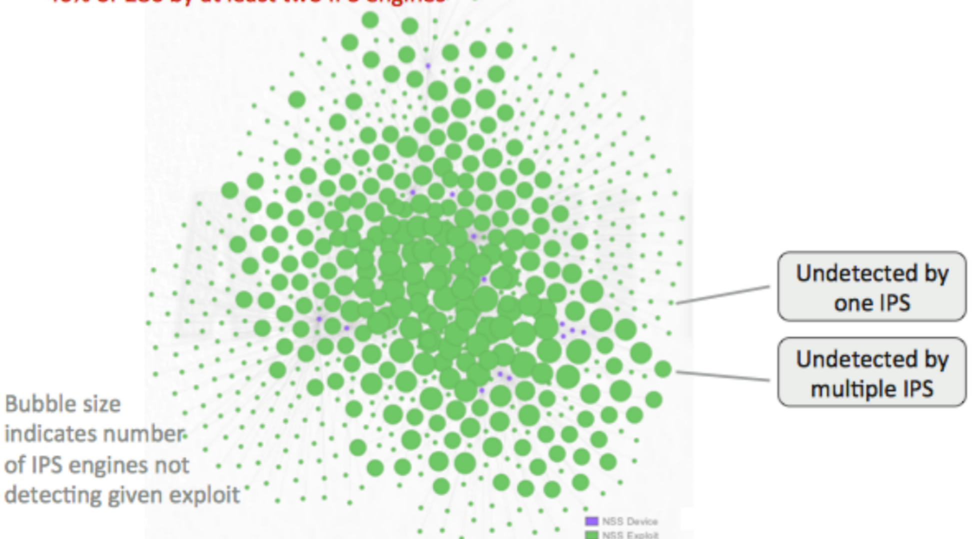
Why AEG?

- Fixing vulnerabilities
  - Need: automated vul assessment
- Vulnerability Assessment
  - Need: assess vulnerability with AEG
- **Intrusion Detection**
  - Malware life becomes shorter
    - FireEye: most malware dies in 2 hours
  - Exploits change frequently
    - NSS : IPS have high false negatives
  - Need: signature generation with AEG



Many exploits are not detected by several IPS engines

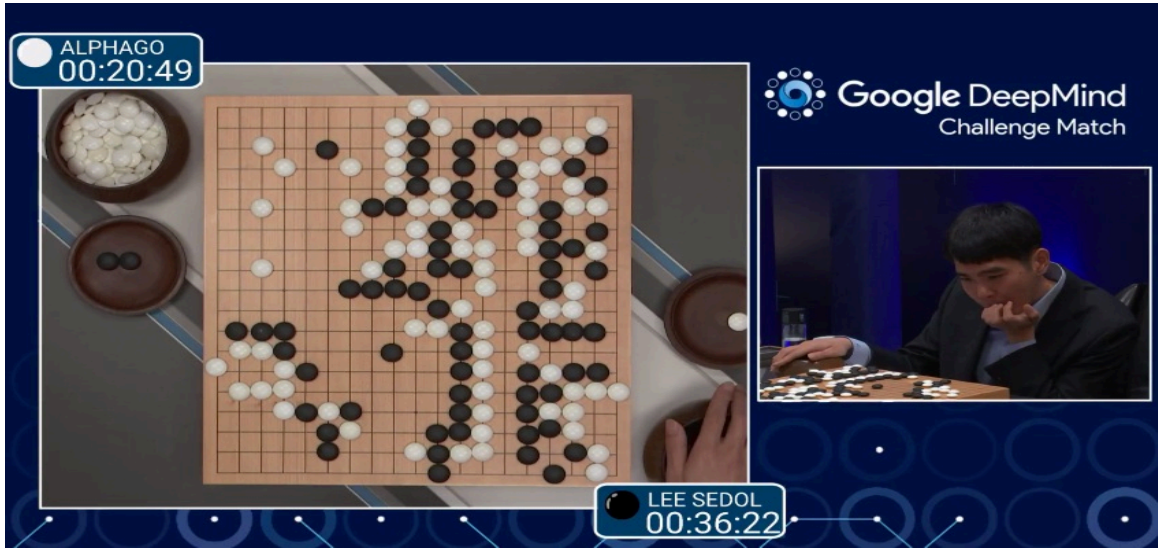
714 of 1,486 exploits tested are not detected by at least one IPS engine, 40% or 286 by at least two IPS engines



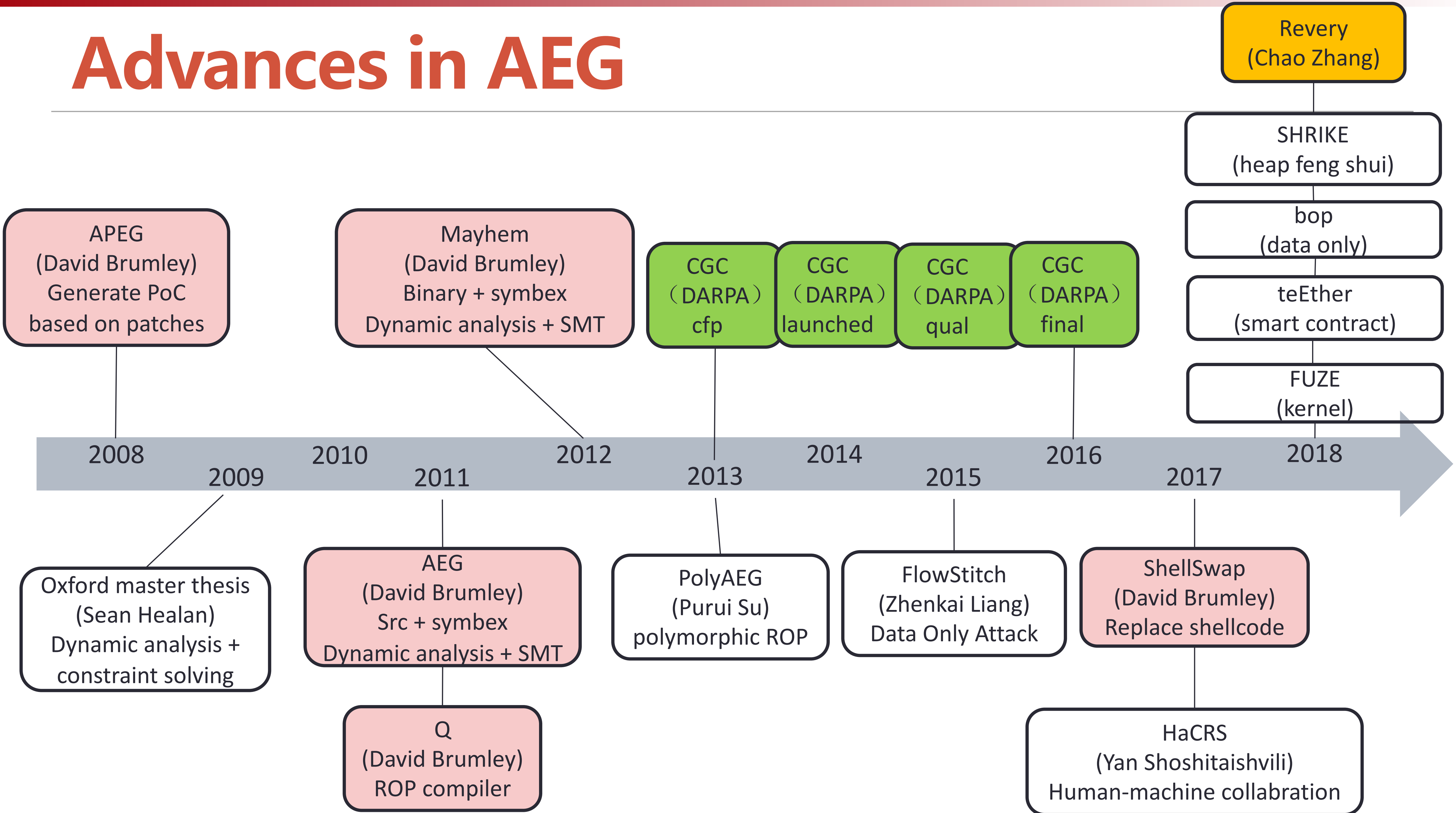
Bubble size indicates number of IPS engines not detecting given exploit

Why AEG?

- Fixing vulnerabilities
  - Need: automated vul assessment
- Vulnerability Assessment
  - Need: assess vulnerability with AEG
- Intrusion Detection
  - Need: signature generation with AEG
- **The practice and trend**
  - Exploiting is challenging
    - Mostly depends on human
  - The machine is rising
    - AlphaGo
  - Need: AEG



Advances in AEG



DARPA Cyber Grand Challenge

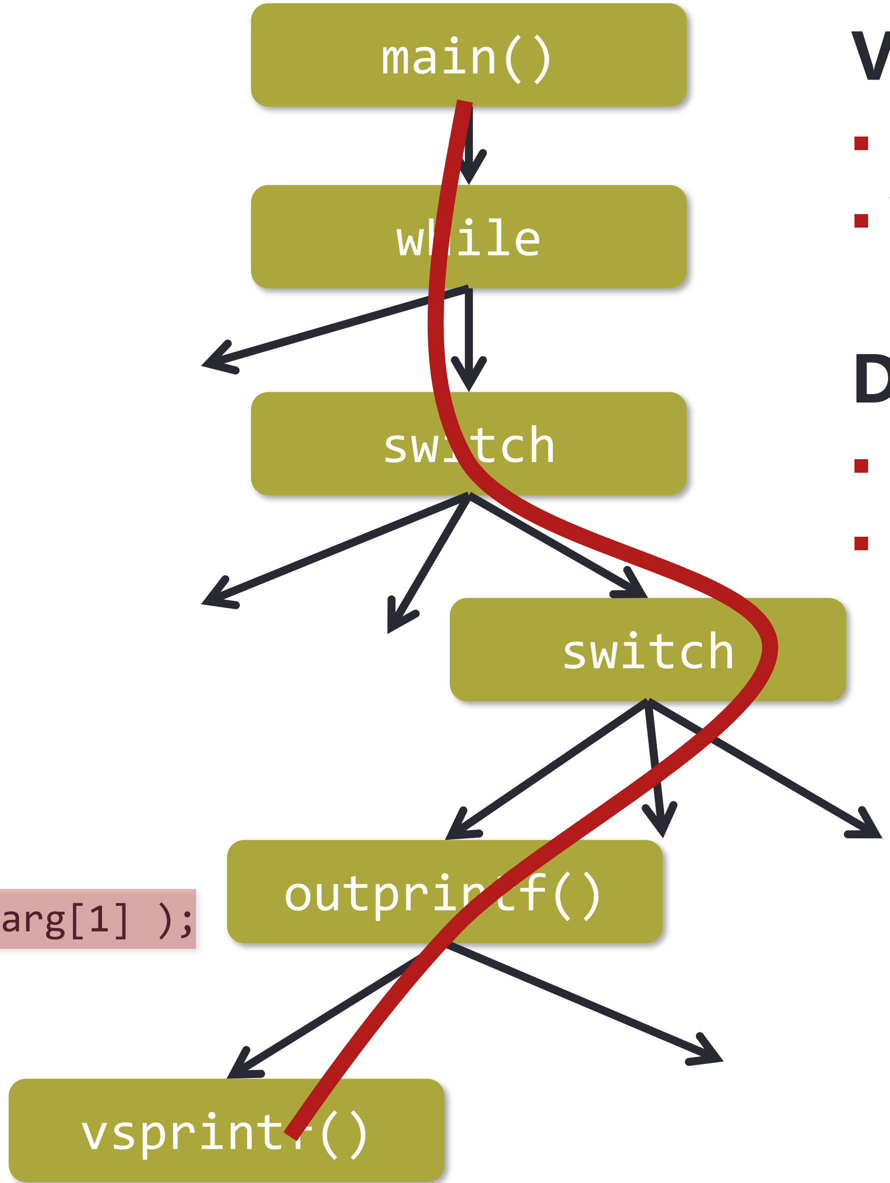
Automated Attack and Defense

（1st in defense in 2015，2nd in offense in 2016）

What is AEG?

Sample Vulnerability : 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: …
    }
  …
```



Vul trigger conditions:

- Path constraints
- Vul constraints

Discover vulnerabilities

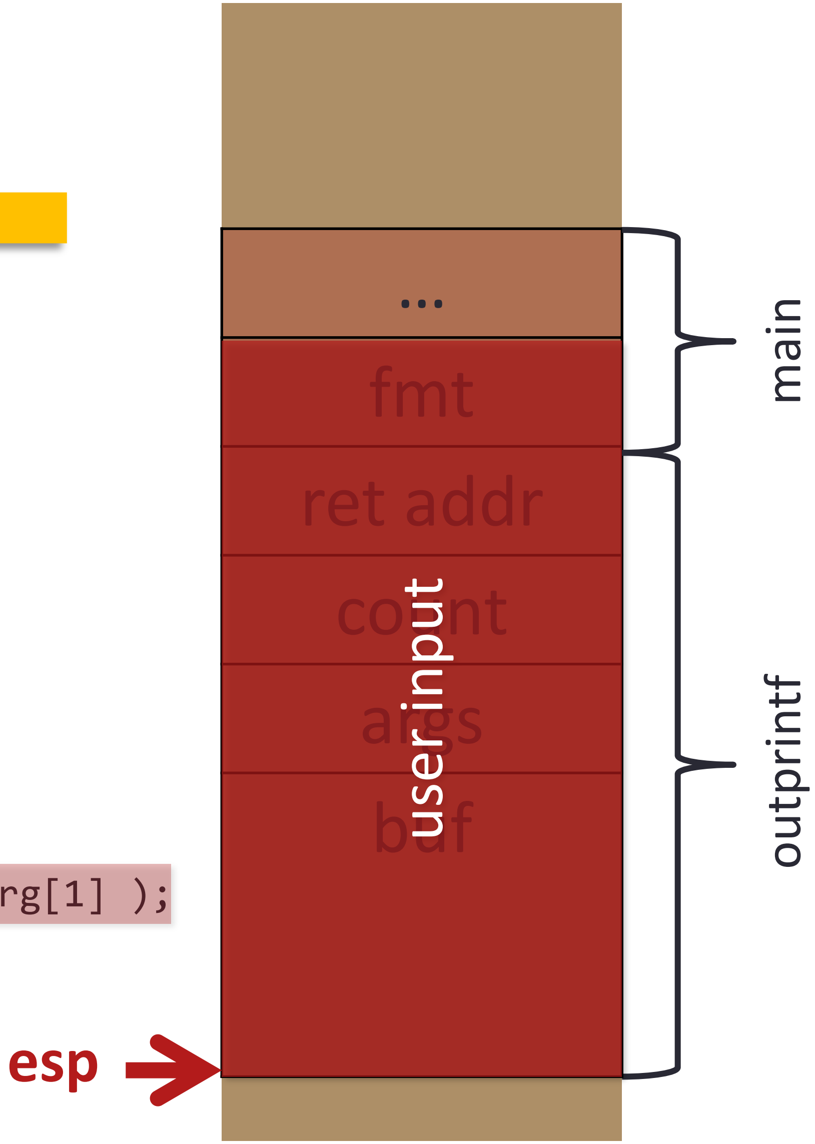
- Symbolic execution
- Fuzzing (testing)

Exploit Vulnerability : 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
}
```

Function returns

```
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: …
    }
…
```



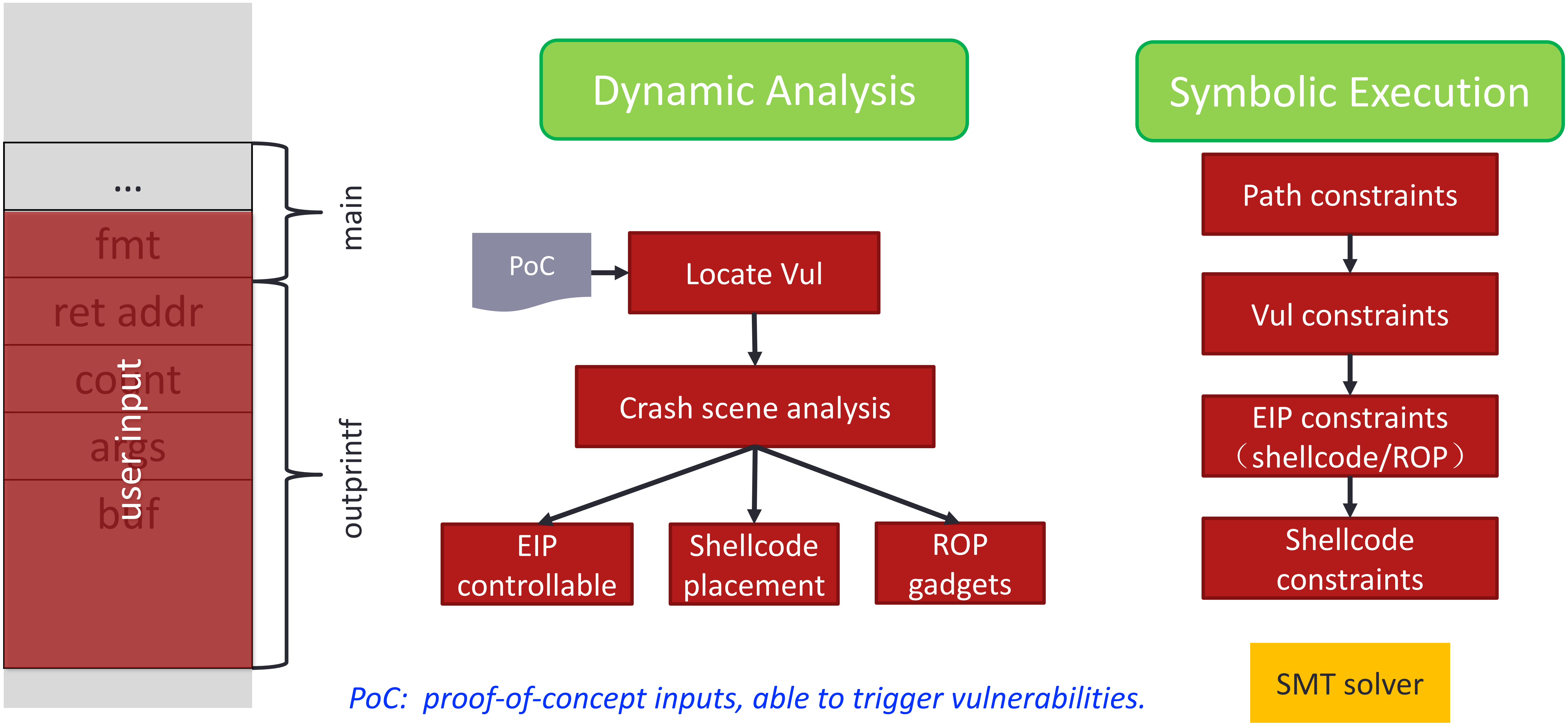
To exploit vul:

- Trigger vul:
  - Path constraints
  - Vul constraints
- Manipulate states:
  - Shellcode constraint
  - EIP constraints
  - Memory layout
  - Defense bypass

Solutions :

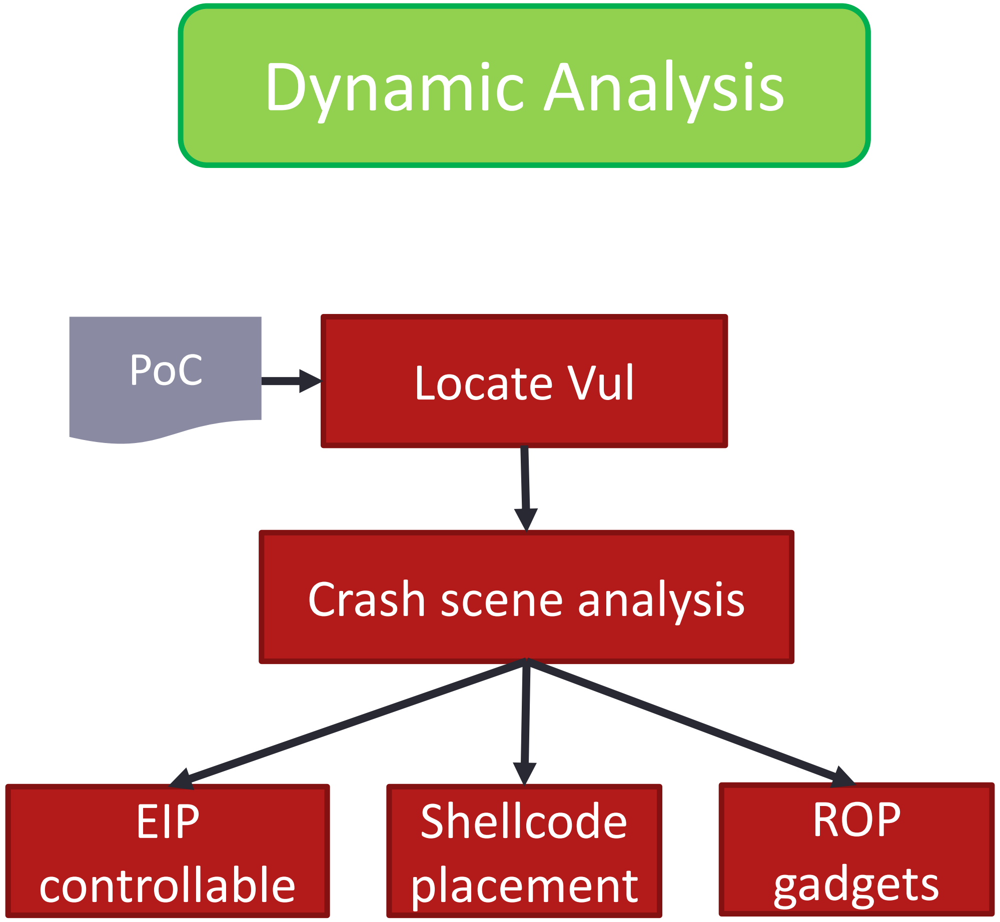
- Symbolic execution

General Workflow of AEG



PoC: proof-of-concept inputs, able to trigger vulnerabilities.

Challenge: non-exploitable PoC

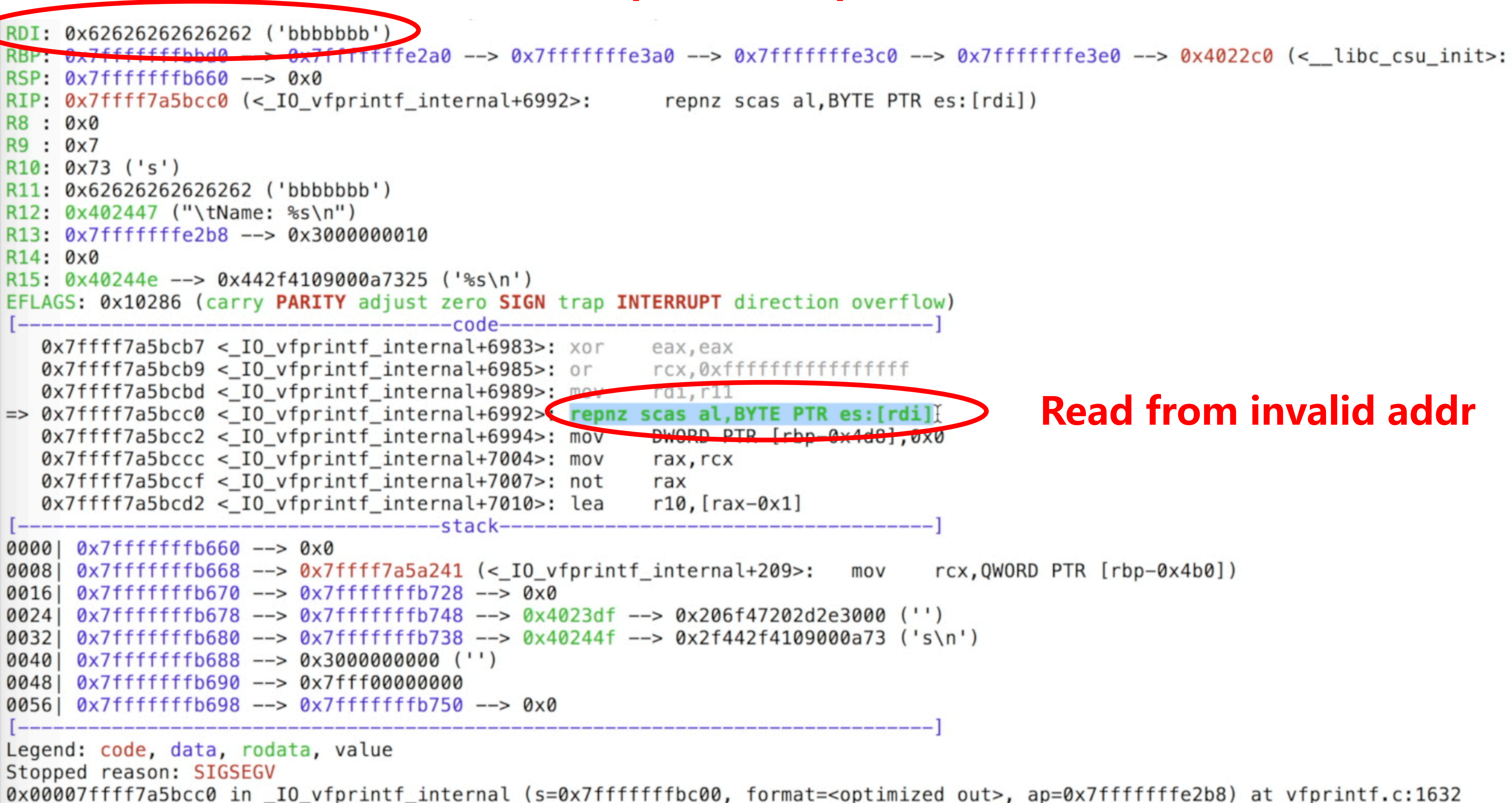


- Sometimes, the PoC is easy to exploit
  - Stack-based buffer overflow
  - Format string vulnerabilities
- Most often, the PoC is non-exploitable
  - The EIP is not controllable
  - The program states cannot be tampered

The crashing path taken by the PoC could be non-exploitable (even by human).

Existing AEG solutions will fail in this case.

Example non-exploitable PoC

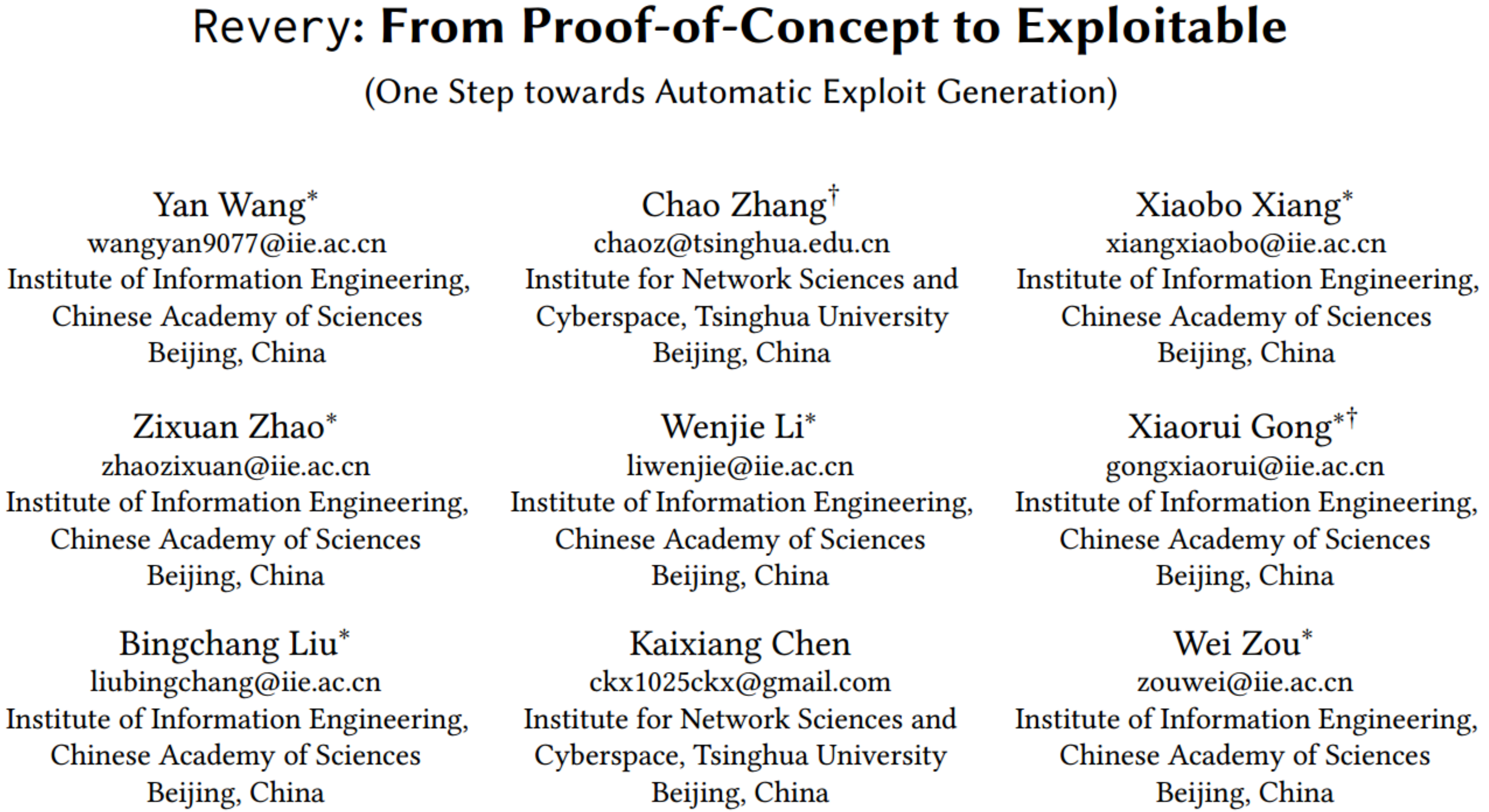


Read from invalid addr

```
RDI: 0x6262626262626262 ('bbbbbbb')
RBP: 0x7fffffffbbd0 --> 0x7fffffffe2a0 --> 0x7fffffffe3a0 --> 0x7fffffffe3c0 --> 0x7fffffffe3e0 --> 0x4022c0 (<__libc_csu_init>:
RSP: 0x7fffffffb660 --> 0x0
RIP: 0x7ffff7a5bcc0 (<_IO_vfprintf_internal+6992>:      repnz scas al,BYTE PTR es:[rdi])
R8 : 0x0
R9 : 0x7
R10: 0x73 ('s')
R11: 0x6262626262626262 ('bbbbbbb')
R12: 0x402447 ("\tName: %s\n")
R13: 0x7fffffffe2b8 --> 0x3000000010
R14: 0x0
R15: 0x40244e --> 0x442f4109000a7325 ('%s\n')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x7ffff7a5bcb7 <_IO_vfprintf_internal+6983>: xor    eax,eax
   0x7ffff7a5bcb9 <_IO_vfprintf_internal+6985>: or     rcx,0xffffffffffffffff
   0x7ffff7a5bcbd <_IO_vfprintf_internal+6989>: mov    rdi,r11
=> 0x7ffff7a5bcc0 <_IO_vfprintf_internal+6992>: repnz scas al,BYTE PTR es:[rdi]
   0x7ffff7a5bcc2 <_IO_vfprintf_internal+6994>: mov    DWORD PTR [rbp-0x4d8],0x0
   0x7ffff7a5bccc <_IO_vfprintf_internal+7004>: mov    rax,rcx
   0x7ffff7a5bccf <_IO_vfprintf_internal+7007>: not    rax
   0x7ffff7a5bcd2 <_IO_vfprintf_internal+7010>: lea    r10,[rax-0x1]
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffb660 --> 0x0
0008| 0x7fffffffb668 --> 0x7ffff7a5a241 (<_IO_vfprintf_internal+209>:   mov    rcx,QWORD PTR [rbp-0x4b0])
0016| 0x7fffffffb670 --> 0x7fffffffb728 --> 0x0
0024| 0x7fffffffb678 --> 0x7fffffffb748 --> 0x4023df --> 0x206f47202d2e3000 ('')
0032| 0x7fffffffb680 --> 0x7fffffffb738 --> 0x40244f --> 0x2f442f4109000a73 ('s\n')
0040| 0x7fffffffb688 --> 0x3000000000 ('')
0048| 0x7fffffffb690 --> 0x7fff00000000
0056| 0x7fffffffb698 --> 0x7fffffffb750 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x00007ffff7a5bcc0 in _IO_vfprintf_internal (s=0x7fffffffbc00, format=<optimized out>, ap=0x7fffffffe2b8) at vfprintf.c:1632
```

The crashing path taken by PoC is non-exploitable.

Look for diverging exploitable paths.



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

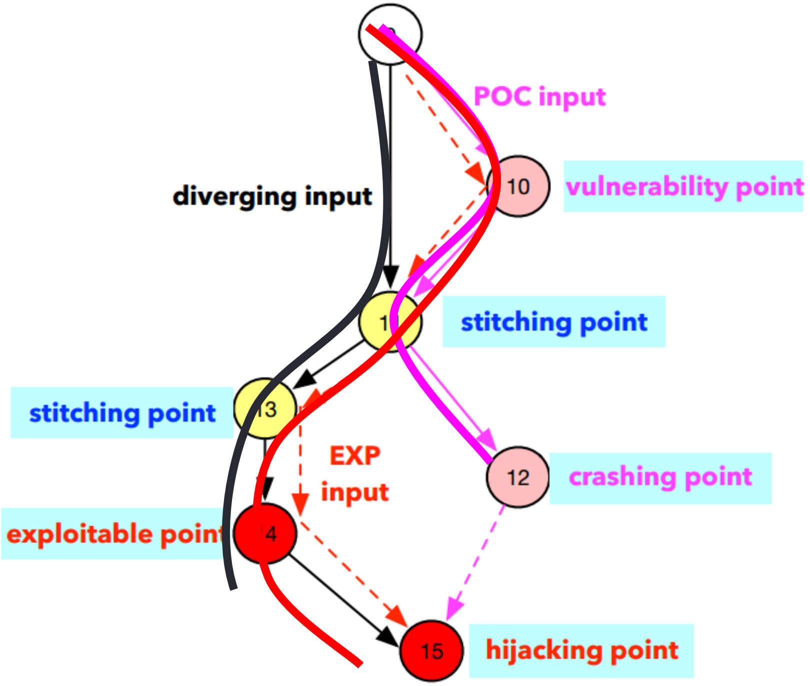
Kaixiang Chen
ckx1025ckx@gmail.com
Institute for Network Sciences and
Cyberspace, Tsinghua University
Beijing, China

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

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. }
```

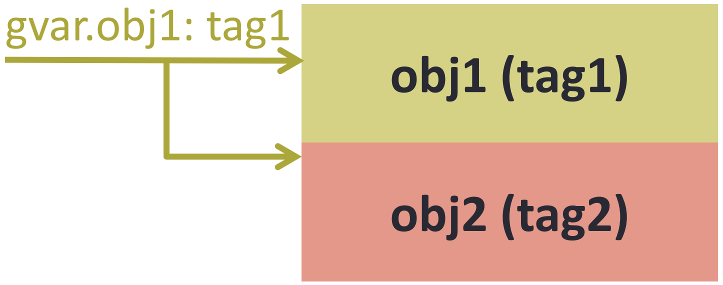


- ➢ **Problem: The crashing path (9->10->11->12->15) taken by PoC is non-exploitable**
- ➢ **Intuition: backtrace the PoC path, look for diverging paths with exploitable states, trigger vul and exploits**
- ➢ **Backtrace to which point?** **How to explore diverging paths?** **How to enter exploitable states?**

Around the vul point **fuzzing (9->11->13->14)** **Path stitching (9-10-11, 11-13-14)**

PoC analysis: locate vulnerability

```
struct Type1 { char[8] data;                          };
struct Type2 { int status;    int* ptr;  void init(){…};  };
int (*handler)(const int*) = …;
struct{Type1* obj1; Type* obj2;} gvar = {};
int foo(){
  gvar.obj1 = new Type1;
  gvar.obj2 = new Type2;
  gvar.obj2->init();  // resulting different statuses
  if(vul)
    scanf("%s", &gvar.obj1->data);    // vulnerability point
  if(gvar.obj2->status)               // stitching point
    res = *gvar.obj2->ptr;            // crashing point
  else                                // stitching point
    *gvar.obj2->ptr = read_int();     // exploitable point
  handler(gvar.obj2->ptr);            // hijacking point
  return res;
}
```

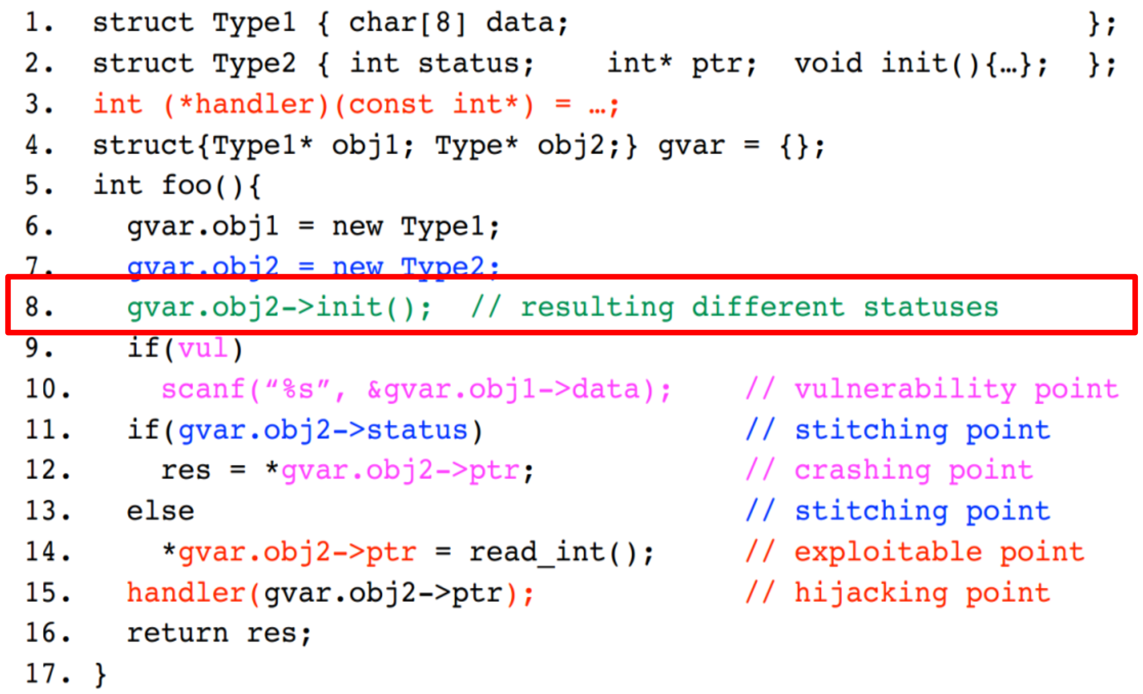


Each object is associated with: a birthmark taint tag, and an access state (uninitialized, busy, freed)

Analyze the PoC's execution trace, and validate the following security rules:

- **V1:** only access objects with intended birthmark, e.g., tag_ptr == tag_obj
- **V2:** only read objects with busy status
- **V3:** only write objects with non-freed status

PoC analysis: identify exceptional object

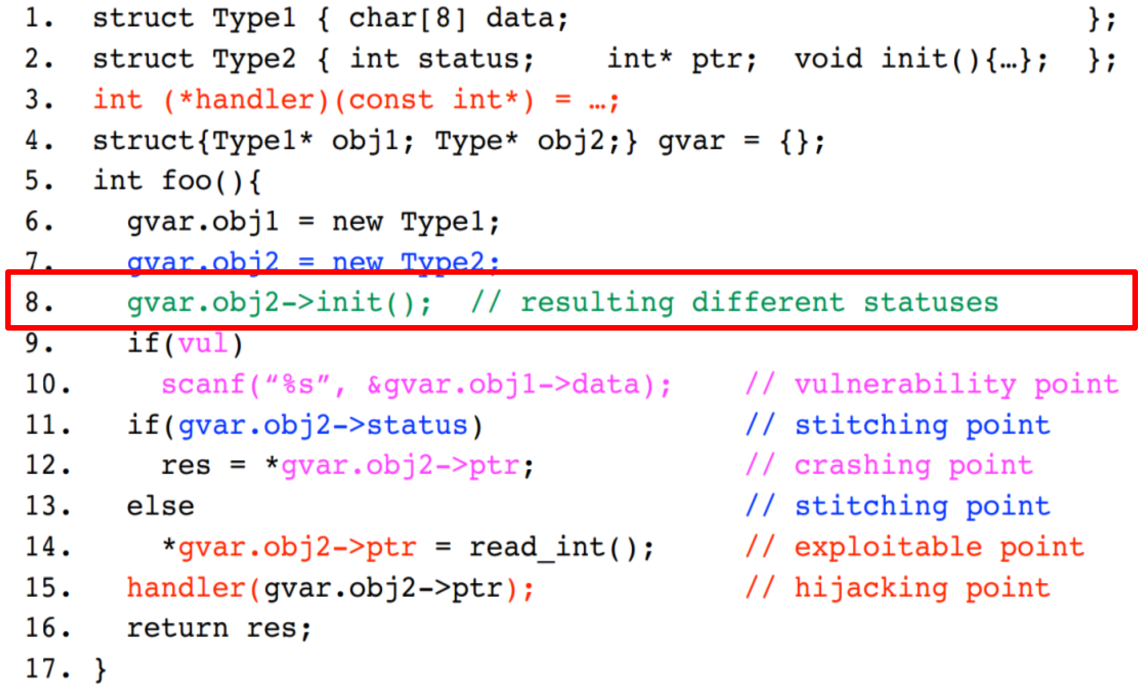


```
struct Type1 { char[8] data;                          };
struct Type2 { int status;     int* ptr;  void init(){…};  };
int (*handler)(const int*) = …;
struct{Type1* obj1; Type* obj2;} gvar = {};
int foo(){
  gvar.obj1 = new Type1;
  gvar.obj2 = new Type2;
  gvar.obj2->init();  // resulting different statuses
  if(vul)
    scanf("%s", &gvar.obj1->data);     // vulnerability point
  if(gvar.obj2->status)                // stitching point
    res = *gvar.obj2->ptr;             // crashing point
  else                                 // stitching point
    *gvar.obj2->ptr = read_int();      // exploitable point
  handler(gvar.obj2->ptr);             // hijacking point
  return res;
}
```

- Exceptional objects: tampered by the vulnerability
  - E.g., obj2 is tampered by the buffer overflow in obj1.

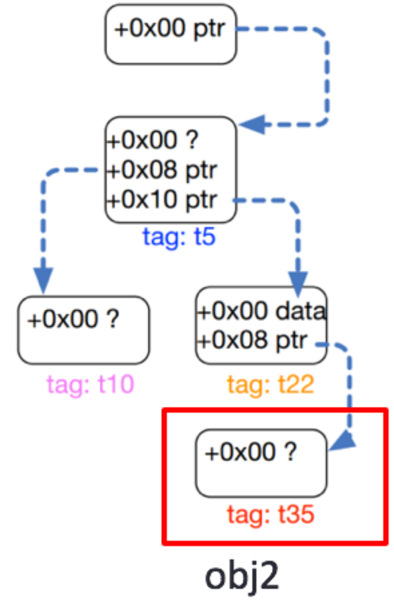
Exceptional objects important, since they are controlled by attackers, and further operations on them could cause programs being exploited.

PoC analysis: identify exceptional object



```
struct Type1 { char[8] data;                             };
struct Type2 { int status;    int* ptr;  void init(){…};  };
int (*handler)(const int*) = …;
struct{Type1* obj1; Type* obj2;} gvar = {};
int foo(){
  gvar.obj1 = new Type1;
  gvar.obj2 = new Type2;
  gvar.obj2->init();  // resulting different statuses
  if(vul)
    scanf(“%s”, &gvar.obj1->data);    // vulnerability point
  if(gvar.obj2->status)               // stitching point
    res = *gvar.obj2->ptr;            // crashing point
  else                                // stitching point
    *gvar.obj2->ptr = read_int();     // exploitable point
  handler(gvar.obj2->ptr);            // hijacking point
  return res;
}
```

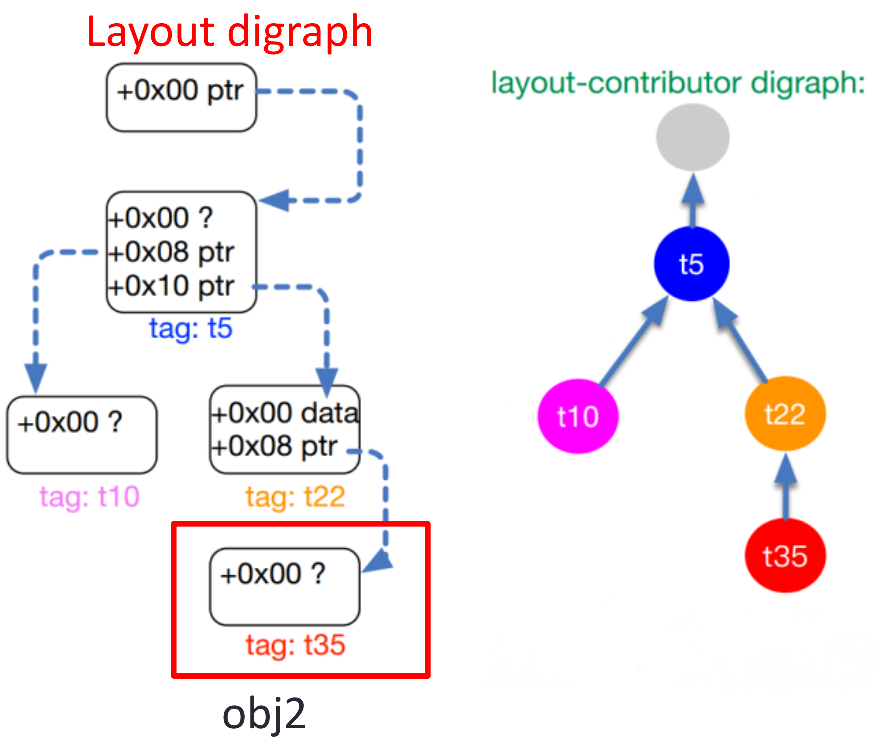
Layout digraph



- ➢ Exceptional objects
- ➢ Exceptional objects’ layout digraph:
  - ➢ **describes how the exceptional object is placed in memory.**
  - ➢ Nodes: memory objects
  - ➢ Edges: Point-to relationship between objects

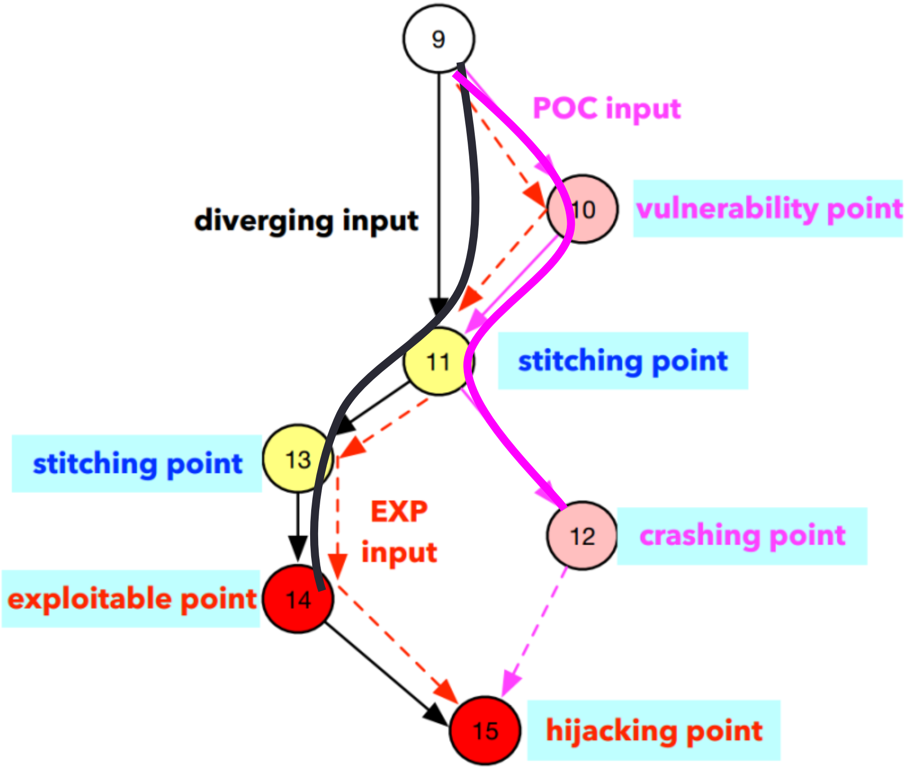
PoC analysis: identify exceptional object

```
struct Type1 { char[8] data;                          };
struct Type2 { int status;    int* ptr;  void init(){…};  };
int (*handler)(const int*) = …;
struct{Type1* obj1; Type* obj2;} gvar = {};
int foo(){
  gvar.obj1 = new Type1;
  gvar.obj2 = new Type2;
  gvar.obj2->init();  // resulting different statuses
  if(vul)
    scanf("%s", &gvar.obj1->data);    // vulnerability point
  if(gvar.obj2->status)               // stitching point
    res = *gvar.obj2->ptr;            // crashing point
  else                                // stitching point
    *gvar.obj2->ptr = read_int();     // exploitable point
  handler(gvar.obj2->ptr);            // hijacking point
  return res;
}
```



- ➢ Exceptional objects
- ➢ Exceptional objects' layout digraph:
- ➢ Exceptional objects' layout-contributor digraph :
  - ➢ **describes how to generate objects similar to exceptional objects.**
  - ➢ Nodes: instructions which allocate the objects
  - ➢ Edges: instructions which setup the point-to relationship between objects

Explore Diverging Paths

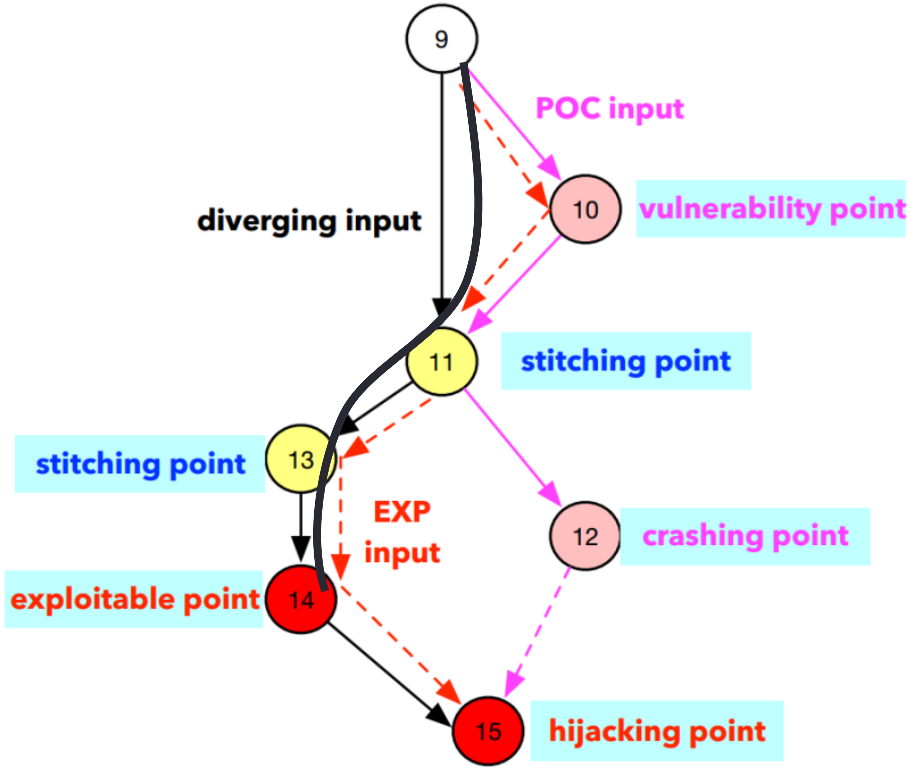


- Backtrace the PoC path to vulnerability points

- Explore diverging paths around vulnerability points
  - Have similar layouts as exceptional objects
    - Could be controlled by attackers
  - Have sensitive operations on those objects
    - Could cause damages to programs

How to explore diverging paths?

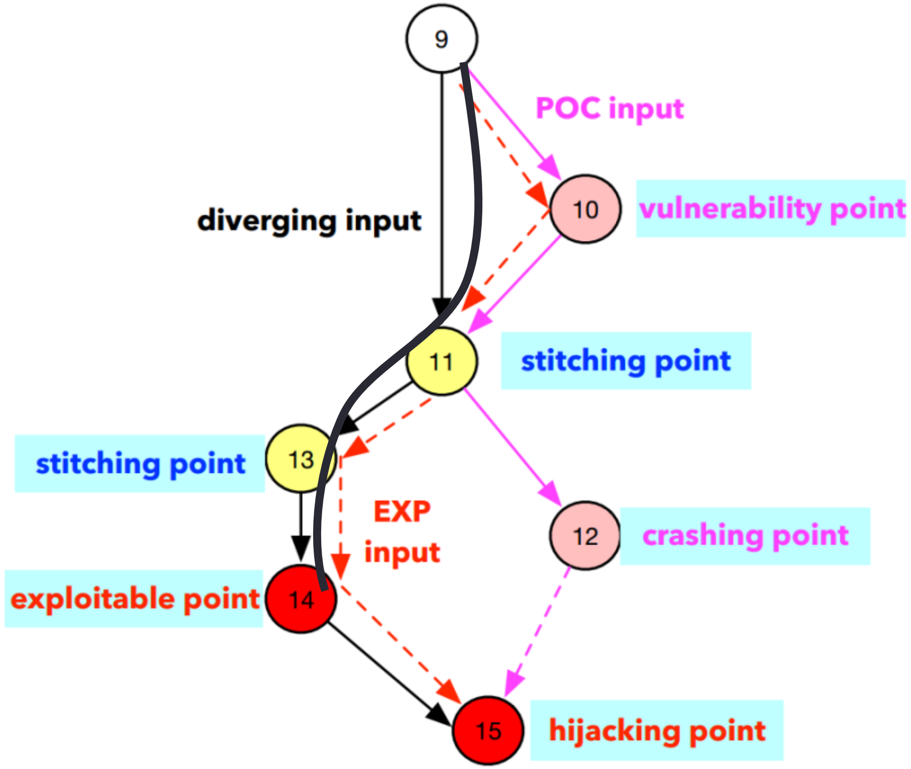
Explore Diverging Paths



How to explore diverging path?

- **A straightforward solution: symbolic execution**
  - Explore program paths symbolically
- But it has scalability issue
  - Path explosion issue
  - Constraint solving challenge
  - Symbolic value concretization
    - Memory allocation: symbolic size
    - Memory access: symbolic index
  - The concrete value could be improper for exploiting.

Explore Diverging Paths

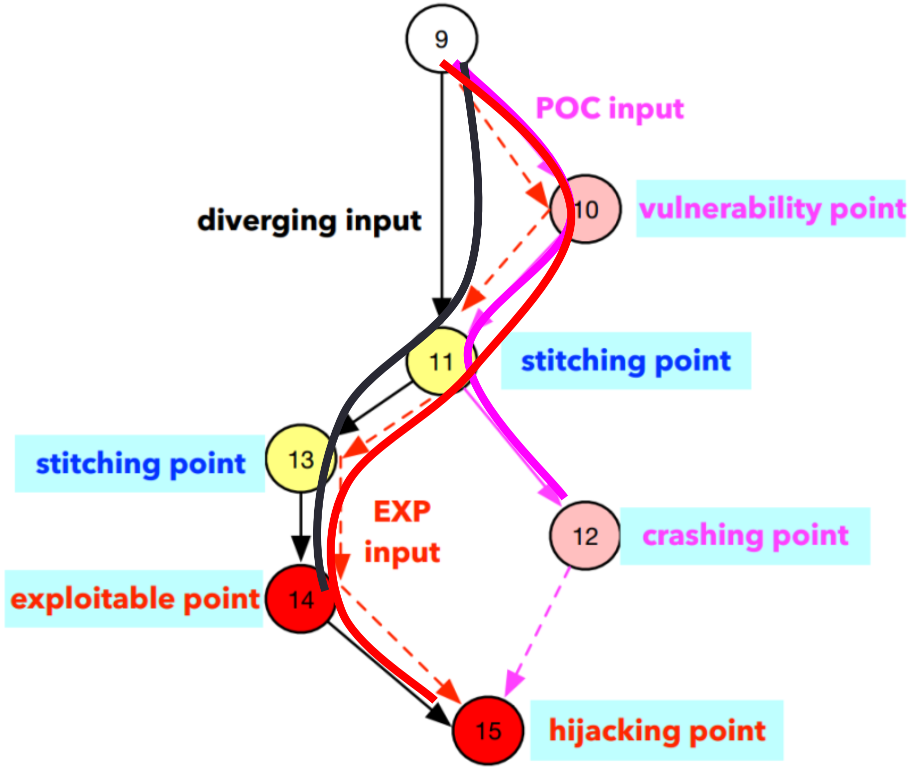


Diverging path: 9->11->13->14
(not necessary to trigger vulnerability)

How to explore diverging path?

- **Our solution: layout-oriented fuzzing**
  - Explore paths by fuzz testing
  - Directed fuzzing: use the memory layout contributor instructions as targets
    - Following these instructions, we can generate objects similar to exceptional objects.
  - Path filtering: find paths that have sensitive operations operating on those objects
    - Exploitable states

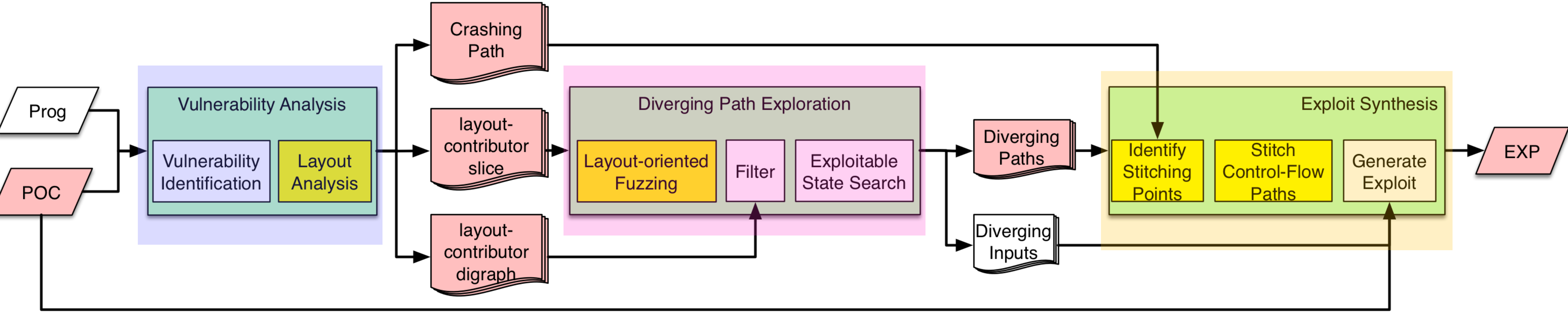
Exploit Synthesis



Crashing path: 9->10->11->12
Diverging path: 9->11->13->14
Exploiting path: 9->10->11->13->14->15

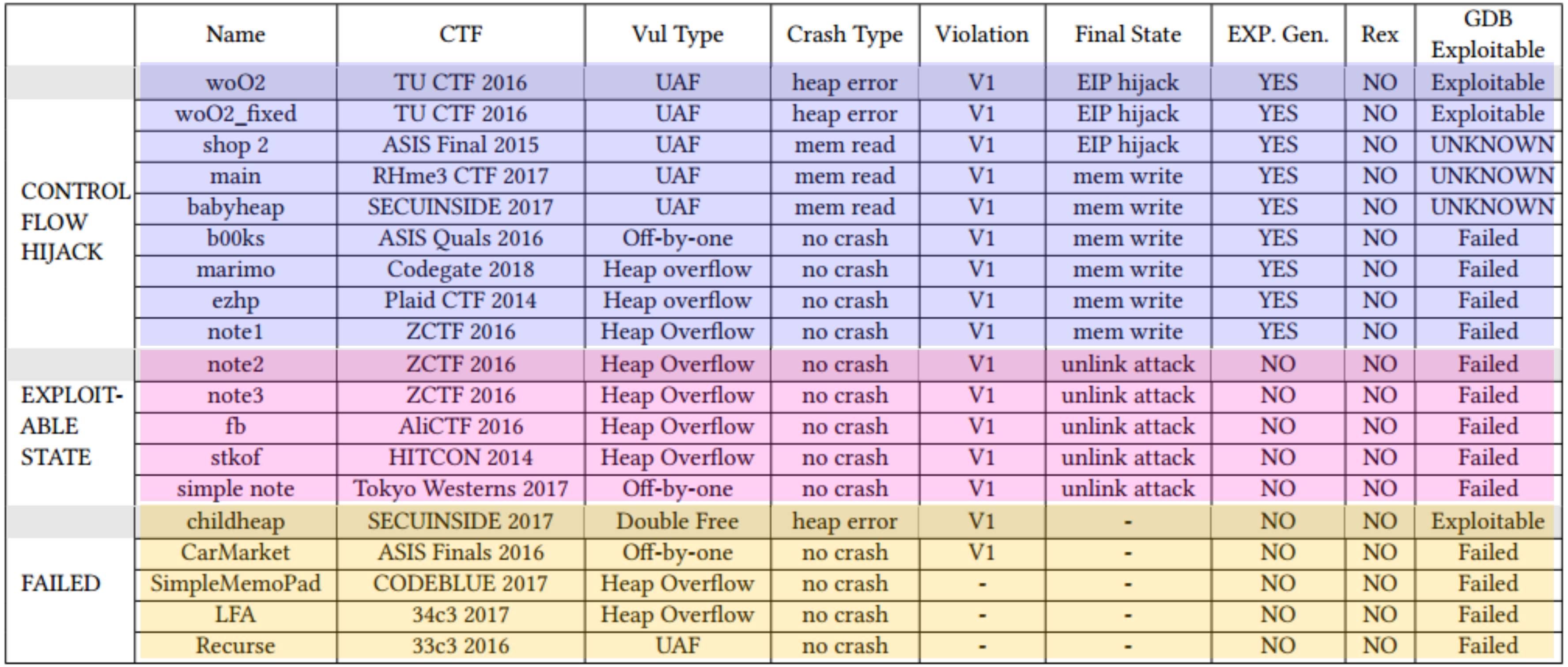
- Find stitching points
  - Try and error
  - Metrics: path reusing rate
- Path stitching
  - Explore candidate sub-paths between stitching points, with symbolic execution
- Exploit generation :
  - Solve the constraints in stitched path
  - Trigger vulnerabilities, and enter exploitable states.

Revery Overview



- ➢ Analyze vulnerability and exceptional objects
- ➢ Explore diverging paths with layout-oriented fuzzing, and find exploitable states
- ➢ Stitch PoC path and diverging path, solve constraints and generate exploits.

Evaluation



	Name	CTF	Vul Type	Crash Type	Violation	Final State	EXP. Gen.	Rex	GDB Exploitable
CONTROL FLOW HIJACK	woO2	TU CTF 2016	UAF	heap error	V1	EIP hijack	YES	NO	Exploitable
	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
EXPLOIT-ABLE STATE	note2	ZCTF 2016	Heap Overflow	no crash	V1	unlink attack	NO	NO	Failed
	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
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
	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

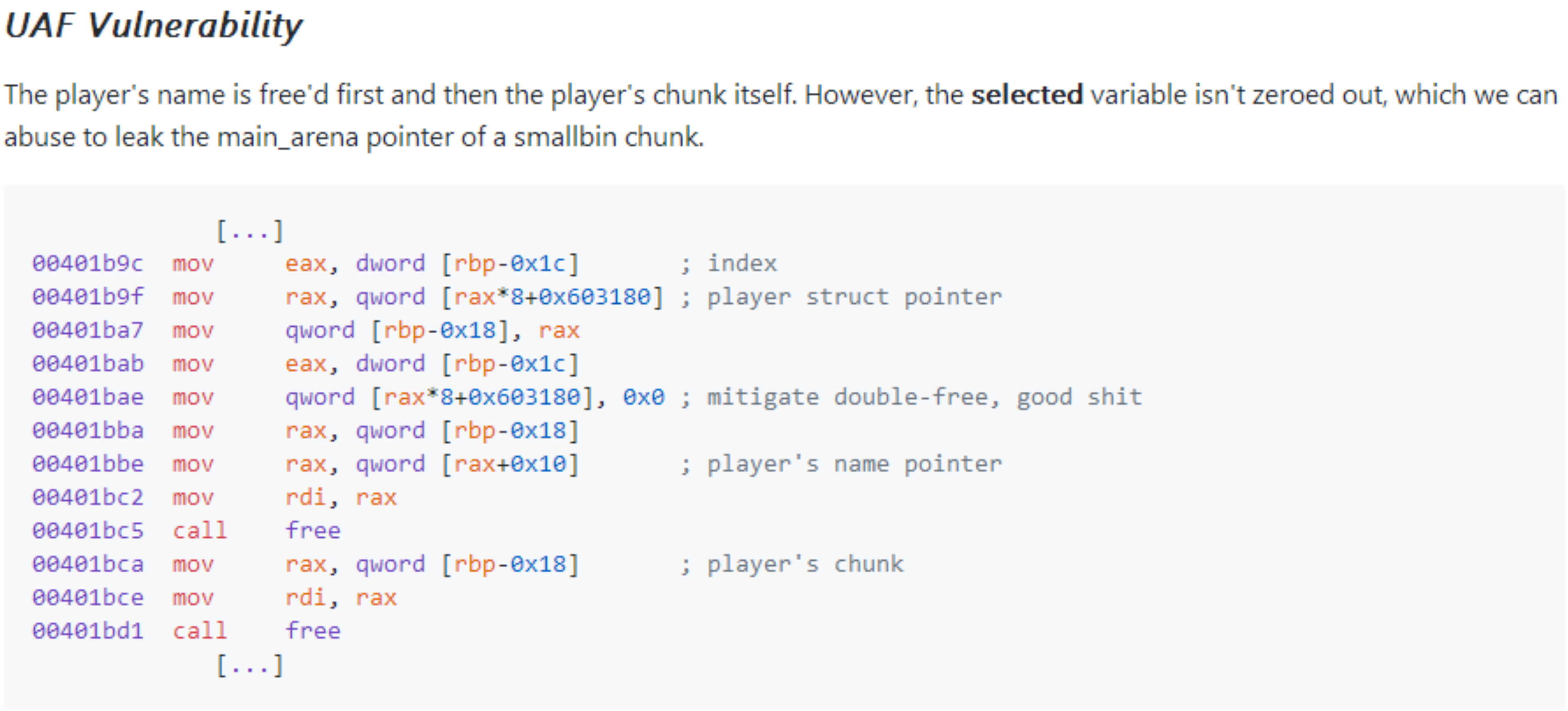
- Target applications: **19** CTF challenges
- Revery generates exploits for **9** of them, triggers exploitable states for **5**, fails for another **5**
- Revery could do AEG for memory read corruption, heap corruption and non-crashing PoC。

Revery's limitation：

- It's based on Angr, lacking support for many syscalls, unable to support real world applications
- It cannot bypass defenses like ASLR yet. So we disable this defense in the evaluation.

DEMO

- A UAF pwn from RHme3 CTF 2017.



UAF Vulnerability

The player's name is free'd first and then the player's chunk itself. However, the **selected** variable isn't zeroed out, which we can abuse to leak the main_arena pointer of a smallbin chunk.

```
               [...]
00401b9c  mov     eax, dword [rbp-0x1c]       ; index
00401b9f  mov     rax, qword [rax*8+0x603180] ; player struct pointer
00401ba7  mov     qword [rbp-0x18], rax
00401bab  mov     eax, dword [rbp-0x1c]
00401bae  mov     qword [rax*8+0x603180], 0x0 ; mitigate double-free, good shit
00401bba  mov     rax, qword [rbp-0x18]
00401bbe  mov     rax, qword [rax+0x10]       ; player's name pointer
00401bc2  mov     rdi, rax
00401bc5  call    free
00401bca  mov     rax, qword [rbp-0x18]       ; player's chunk
00401bce  mov     rdi, rax
00401bd1  call    free
               [...]
```

- Generate an exploit for a PoC crashing at a non-exploitable memory read operation.
  Note: The ASLR defense is turned off in the experiment.

Takeaway

AEG vs. Revery

- Traditional AEG solutions:
  - Highly depend on the crashing scene
  - Use dynamic analysis and symbex
- Challenges
  - PoC crashing scene is non-exploitable
  - Symbolic execution is not scalable
  - Poor support for heap vulnerabilities

- Revery
  - Explore diverging paths rather than PoC path
  - Use fuzzing rather than symbolic execution, to explore diverging paths
  - Use layout contributor instructions as targets, to direct the fuzzing and speedup.
  - Use symbolic execution to stitch PoC path and diverging path, to generate exploits

Revery only pushes AEG one small step forward.

Roadblocks of AEG

- Exploit specification (AH)
  - Conditions of anti-specification
  - Partitioning of code privilege
- Exploit generation (BCDE)
  - Infer pre/post conditions
  - Infer loop pre condition
  - Infer paths reachable to targets
  - Exploit derivability (Revery)
- Multi-interaction (F)
  - Multiple vulnerabilities, multi-operations
- Environment manipulation (GIJK)
  - Race condition
  - Memory/heap fengshui
  - Time analysis, to infer size etc.
  - Information leakage, e.g., side channel

The Automated Exploitation Grand Challenge

Tales of Weird Machines

Julien Vanegue

julien.vanegue@gmail.com

H2HC conference, Sao Paulo, Brazil
October 2013

- Infrastructure
  - Symbex, taint analysis, binary analysis…

Thanks!

Q&A