EVERYONE HAS HIS OR HER OWN FUZZER

BEIST (BEISTLAB/GRAYHASH)

WWW.CODEENGN.COM
CODEENGN REVERSEENGINEERING CONFERENCE

2012 CodeEngn

About me

- From beistlab
- Principal research consultant at *GrayHash*
- ***BoB*** mentor

- Over 10 win/run hacking contests (5 times defcon CTF quals)
- ZDI RECON TIPPINGPOINT win /iDefense bounty
- Consulting for big companies
- Hunting bugs is my favorite hobby

About this talk

- 70% = Fuzzing
- 30% = Word for young students

What is fuzzing

- Software testing technic
- Automated or semi-automated
- Security field uses this for finding 0 days
- Mostly for hunting mem-corruption style
- Many bugs come from this way these days

Fuzzing popular?

- Indeed! Security conference always has fuzzing talks these days.
- No strong evidence but you could picture that over 50% bugs are from fuzzing

2 fuzzing ways

- Dumb fuzzing
- Smart fuzzing

Dumb fuzzing

- Random mutation and run cases
- You can do this within only 10 line python

(But dumb fuzzing still works)

```
import os, sys, random
def go():
  return random.randrange(0, 0x100)
filesize = os.path.getsize("./sample.xxx")
fp = open("./sample.xxx", "rb++")
tmpoffset = random.randrange(0, filesize)
fp.seek(tmpoffset, 0)
fp.write("%c%c%c%c" % (go(), go(), go(), go()))
fp.close()
os.system("target_binary sample.xxx")
```

Smart fuzzing

- In security field, something called smart fuzzing if it's not just dumb fuzzing
- Basically no clear definition for that
- But let's tour to see what smart fuzzing people do

What you need to do smart fuzzing

- No specific skills required
- But there are some general technics
- Each of them is very simple but it's hard when you put it all together

Intermediate Language

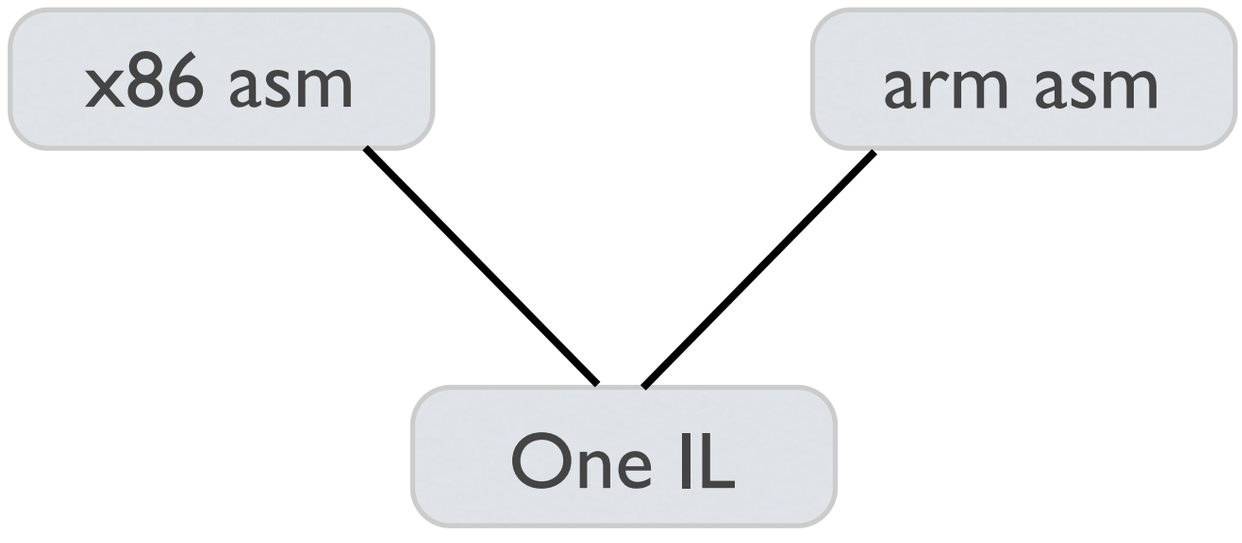
- Not necessary but this reduces the complexity of analysis a lot
- Done by a simpler intermediate language engine for assembly language
- The key is
  - to make it more simpler
  - to make it portable

Intermediate Language

- Many semantics per intel instruction
- How is the complexity of "*inc*" instruction
- Exceptions - 5 conditions
  - GP(0), SS(0), PF(fault-code), AC(0), UD
- Side effects - OF, SF, ZF, AF, PF
- Very useful for static analysis

Intermediate Language

- Many CPU platforms (PC, Phones ++)
- x86, x64, mips, arm, powerpc
- intel, arm, and so on can be represented



- Reference: Zynamics's REIL

Intermediate Language

```
00401F10    slrundll.exe::sub_401F10
00401F9F    mov         edi, ds:[ebx+0x8]
00401FA2    lea         ecx, ds:[esi+esi*0x2]
00401FA5    mov         esi, ds:[edi+ecx*0x4]
00401FA8    jmp         loc_401F4B
```

```
401F9F00    add         0x8, ebx, qword t0        // 00401F9F mov edi, ds: [ebx + 8]
401F9F01    and         qword t0, 0xFFFFFFFF, t1
401F9F02    ldm         t1,   , t2
401F9F03    str         t2,   , edi
401FA200    mul         0x2, esi, qword t0        // 00401FA2 lea ecx, ds: [esi + esi * 2]
401FA201    add         qword t0, esi, qword t2
401FA202    and         qword t2, 0xFFFFFFFF, t3
401FA203    str         t3,   , ecx
401FA500    mul         0x4, ecx, qword t0        // 00401FA5 mov esi, ds: [edi + ecx * 4]
401FA501    add         qword t0, edi, qword t2
401FA502    and         qword t2, 0xFFFFFFFF, t3
401FA503    ldm         t3,   , t4
401FA504    str         t4,   , esi
401FA800    jcc         byte 0x1,   , 0x401F4B    // 00401FA8 jmp loc_401F4B
```

[x86 to REIL]

Symbolic Execution

- Good money example

```
if (a==1337)
    strcpy(a,b);      <-- {block 1}
else
    printf("x");      <-- {block 2}
```

- How do you get into *block 1* to call strcpy() function?

Symbolic Execution

```
func(a, b):
  a = a + b
  b = b + b
  a = a - b
if a == 2:
  strcpy()
else:
  exit
```

[code]

a, b

$a_1 = a_0 + b_0$

$b_1 = b_0 + b_0$

$a_2 = a_1 - b_1$

$if\ a_2 == 2:$
  strcpy

else:
  exit

[symbolic expression]

Symbolic Execution

- Then we pass symbolic expression to SMT (or something else)
- To know the condition to get into the path
- Security field uses SAT/SMT to do code coverage and find vulnerabilities
- Academy meet-up also has those topics
- Reference: SAT/SMT summer school, SAGE(MS), CMU's AEG

Taint analysis

- Usually to track register or data that influences others
- "e*dx*" will be user-controllable after "*pop* e*dx*"
- We want to know if we can control "e*cx*"

```
pop edx
pop ebx
add ebx, edx
mov eax, ebx
mov ecx, [eax]
rep movsd
```

Generating cases based on file specification

- If there is a file specification, hackers can reference it to generate better cases
- Example:

UINT16	2	속성(표 13 참조)
WORD	2	Type이 "LINK"일 때, 연결 파일의 절대 경로 길이 (len1)
WCHAR array[len1]	2×len1	Type이 "LINK"일 때, 연결 파일의 절대 경로

[HWP specification]

Generating cases based on whitebox fuzzing

- In systematic dynamic test generation
- Fuzzer tracks down its flow and get useful information while dynamic program analysis
- This can be done by symbolic execution / taint analysis

Generating cases based on whitebox fuzzing

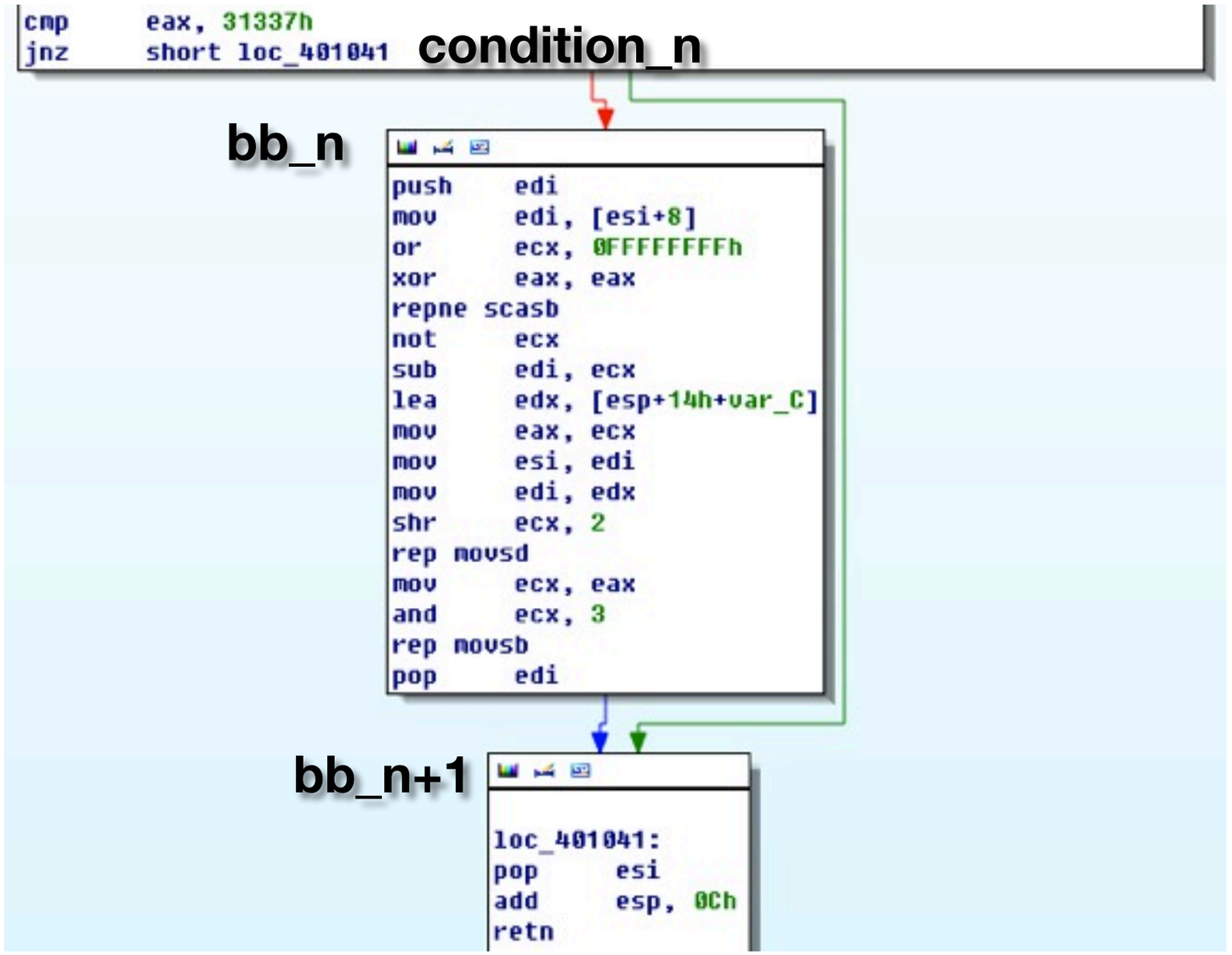
- This is a very tough challenge
- Constraint solving could be imprecise due to the complexity
  - floating-point operations
  - Sys calls
  - etc
- References: MS's SAGE

Comparing flows

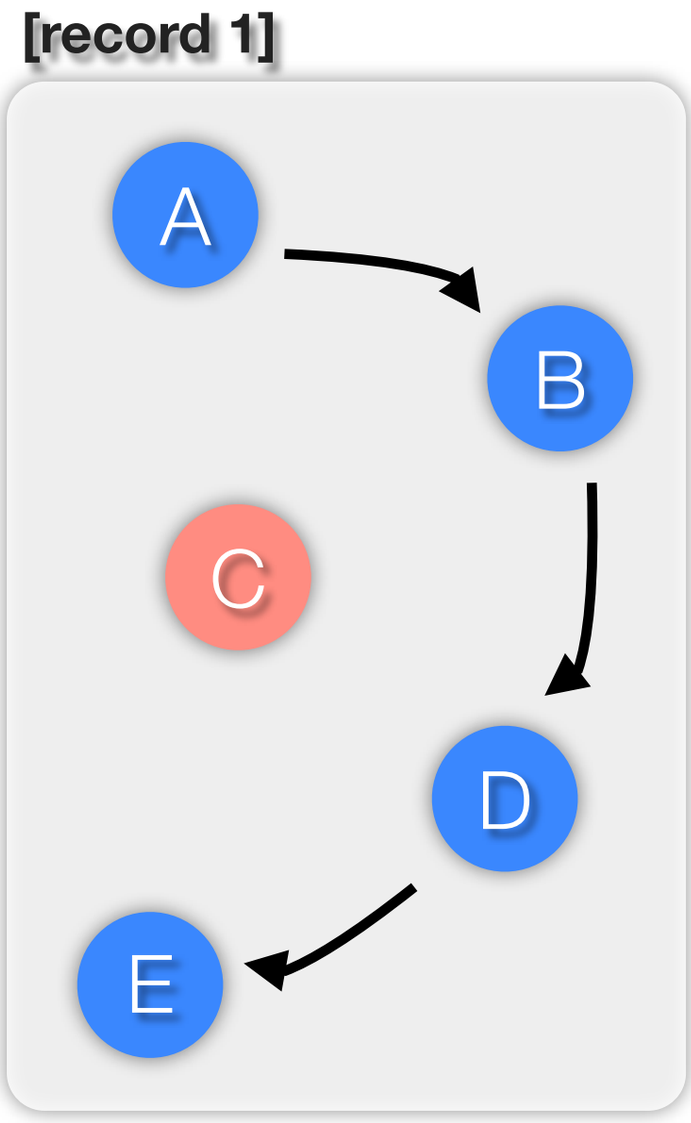
- You run 2 file cases and record flows
- And compare the 2 flows
- Figure out the difference
- which position of file data is used by target program

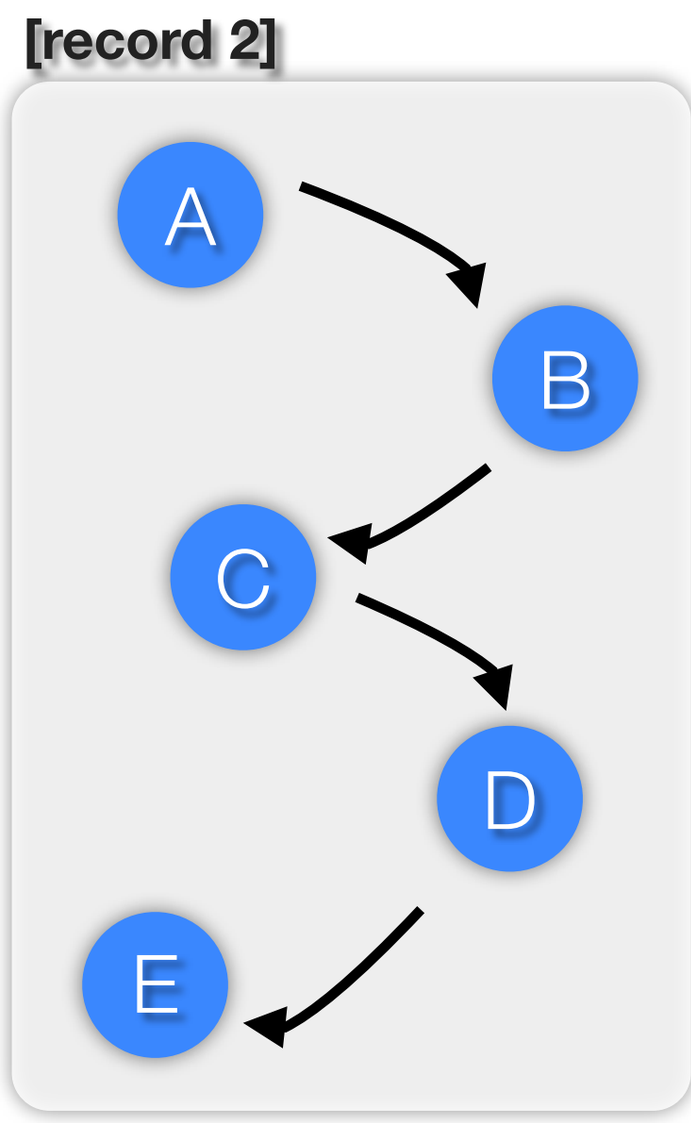
Comparing flows

- Assume the flow gets to *bb_n*+1
- Find *condition_n*
- Evaluate how we can get into *bb_n*
- Figure out we have to change *EAX* reg
  (Changing eflags directly would have many falses)



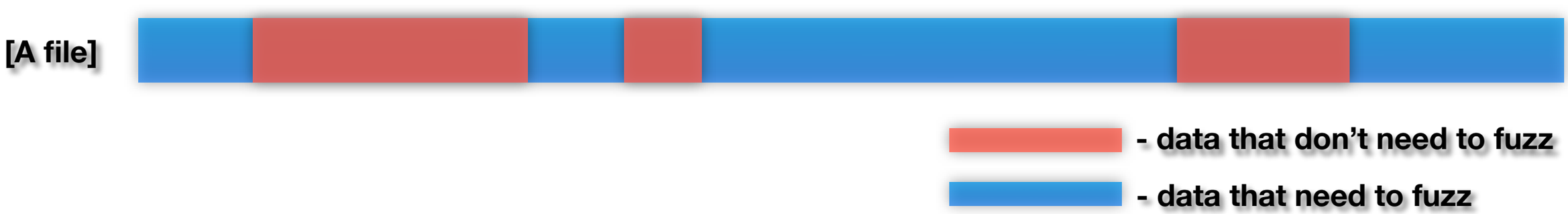
Comparing flows





Comparing flows

- More advantage: you can know which position is actual affecting data
- You don’t have to spend a huge time to fuzz useless data



Blackbox fuzzing but smart

- 0 knowledge fuzzing
- But it's not super random
- This recognizes type of data and gives them some special sauce
- Radamsm: Collection of model-inferred-based fuzzers with some elements

Blackbox fuzzing but smart (Radamsa)

grafu: Mutate a grammar inferred from the data.
fubwt: Mutate the with a disturbed Burrows-Wheeler transformation.
permu: Permute content around the edges of common substrings.
rambo: A traditional file fuzzer making simple changes to raw data.
enumr: Enumerate all byte sequences after a few shared prefixes.
stutr: Add repetitions to the data.
tehfu: Build a tree using simple rules and mutate it.
cutup: Splice and permute the contents of the files.
flipr: A bit flipper.
walkr: Make systematically mutations to all positions of all files.
range: generate chunks of random data
noise: mix sample data and noise signals together
forml: generate data from random formal languages
proby: Fill holes from files by using statistics.
surfy: Jump between locations with common data.
small: Test case minimizer. Hopefully not effective if used as a fuzzer.

[radamsa modules]

Blackbox fuzzing but smart

- Seems lower effort required if we compare to other smart fuzzers, but

- [$10,000] [116661] **Rockstar** CVE-1337-d00d1: Excessive WebKit fuzzing. *Credit to miaubiz.*
- [$10,000] [116662] **Legend** CVE-1337-d00d2: Awesome variety of fuzz targets. *Credit to Aki Helin of OUSPG.*
- [$10,000] [116663] **Superhero** CVE-1337-d00d3: Significant pain inflicted upon SVG. *Credit to Arthur Gerkis.*

CVE-2011-1434 Chrome, CVE-2010-0001 gzip,
CVE-2010-0192 Adobe, CVE-2011-0155 WebKit,
CVE-2011-0074 Firefox, CVE-2011-0075 Firefox,
CVE-2010-1205 LibPNG,CVE-2010-1793 WebKit,
CVE-2010-1404 WebKit, CVE-2010-1410 WebKit,
CVE-2010-1415 WebKit, CVE-2010-1415 WebKit,
CVE-2011-1186 Chrome,CVE-2011-2348 Chrome

Distributed fuzzing

- There are firms who spend $$$ for distributed fuzzing
- Huge time needed to develop smart fuzzers
- Imagine $100,000 smart fuzzer against $100,000 many distributed fuzzing
- Not-bad-code-coverage without crazy crazy effort = DEAL!

Runtime function fuzzing

- RTL function is everywhere
- Super fast fuzzing if functions don't have any system API (interrupt/sysenter/etc)
- What if you use GPU for RTL func fuzzing?
- Challenge:
  - Recognizing arguments and their types
  - If possible to hit the RTL function
  - If possible to manipulate values of arguments

Runtime function fuzzing

- void test(char *str)
- ***str*** needs a string pointer not int or smth
- Type inference needs: guess how hex-rays displays variable types?
- Think: if a RTL func has non-RTL functions, is it still a RTL func?

USB fuzzing

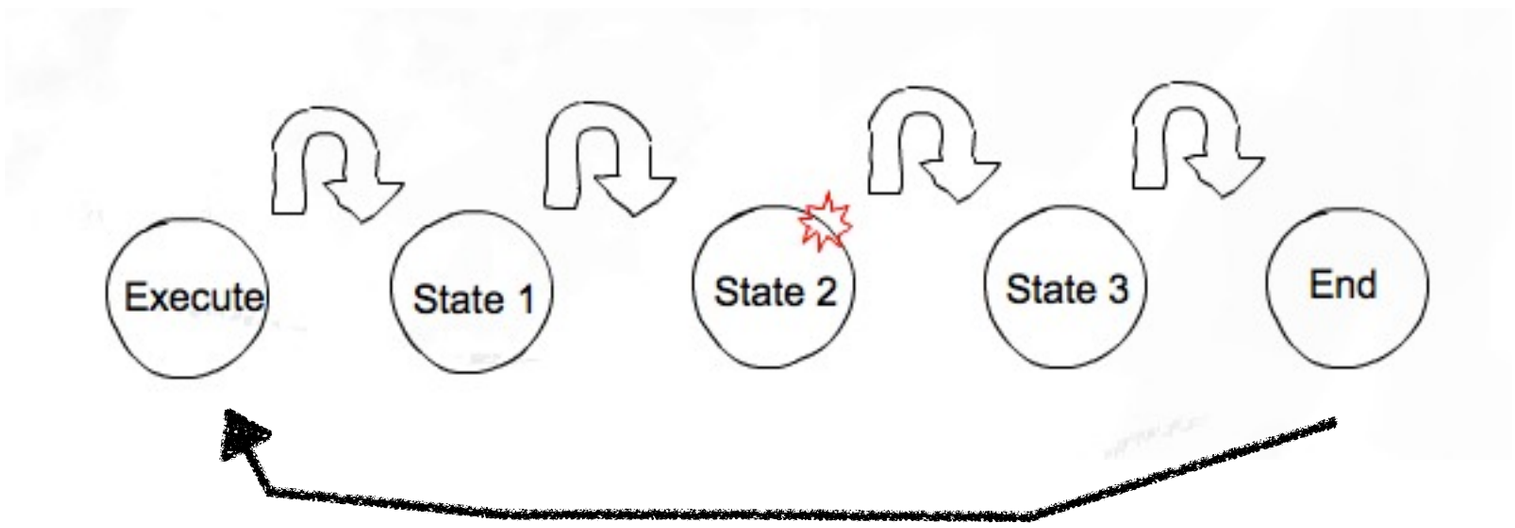
- We can be *007*
- There is a locked computer
- Put your usb into the computer and it pops up calc.exe
- How? Abuse USB protocol or NTFS bugs

Filesystem fuzzing

- NTFS had (or still has) a lot of problems
- You may have kernel BSOD in 5 mins
- No need to fuzz physically but use a mounting program
- Reference: google '*usb or filesystem fuzzing*'

Memory snapshot fuzzing

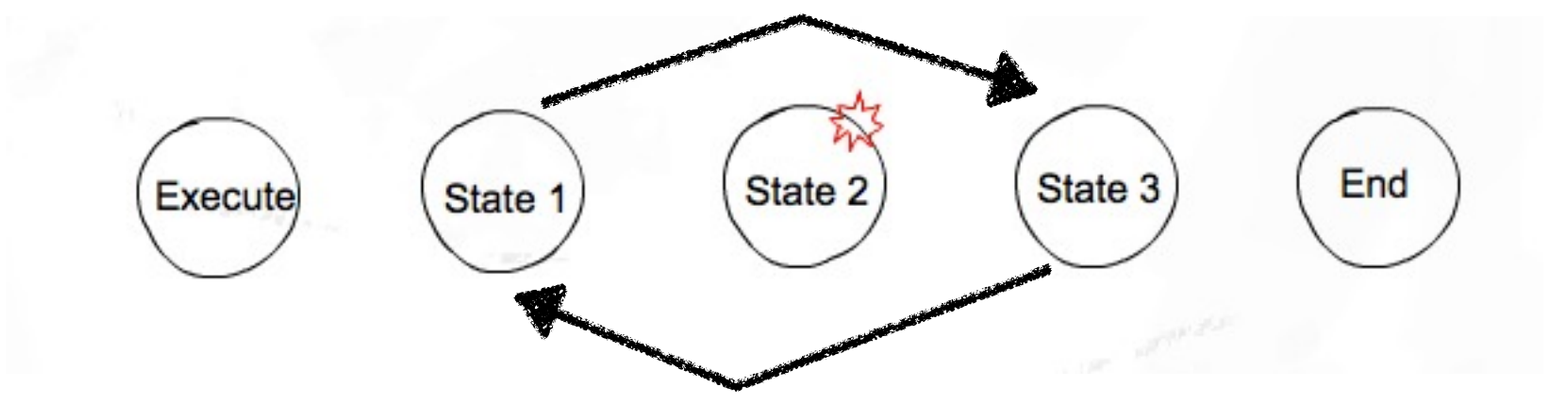
- Motivation: to speed up fuzzing
- Example) The red one is the target to fuzz



[normal fuzzing]

Memory snapshot fuzzing

- Snapshot and fuzzing
- Much more faster
- Challenge: Recover File handle/syscall/external data



[snapshot fuzzing]

Hunting interesting functions

- Modern programs have countless functions
- If you can know which function is interesting, your life would be easier
- Helpful both watching and fuzzing

Hunting interesting functions

- No argument (Except global variables used)
- No loop

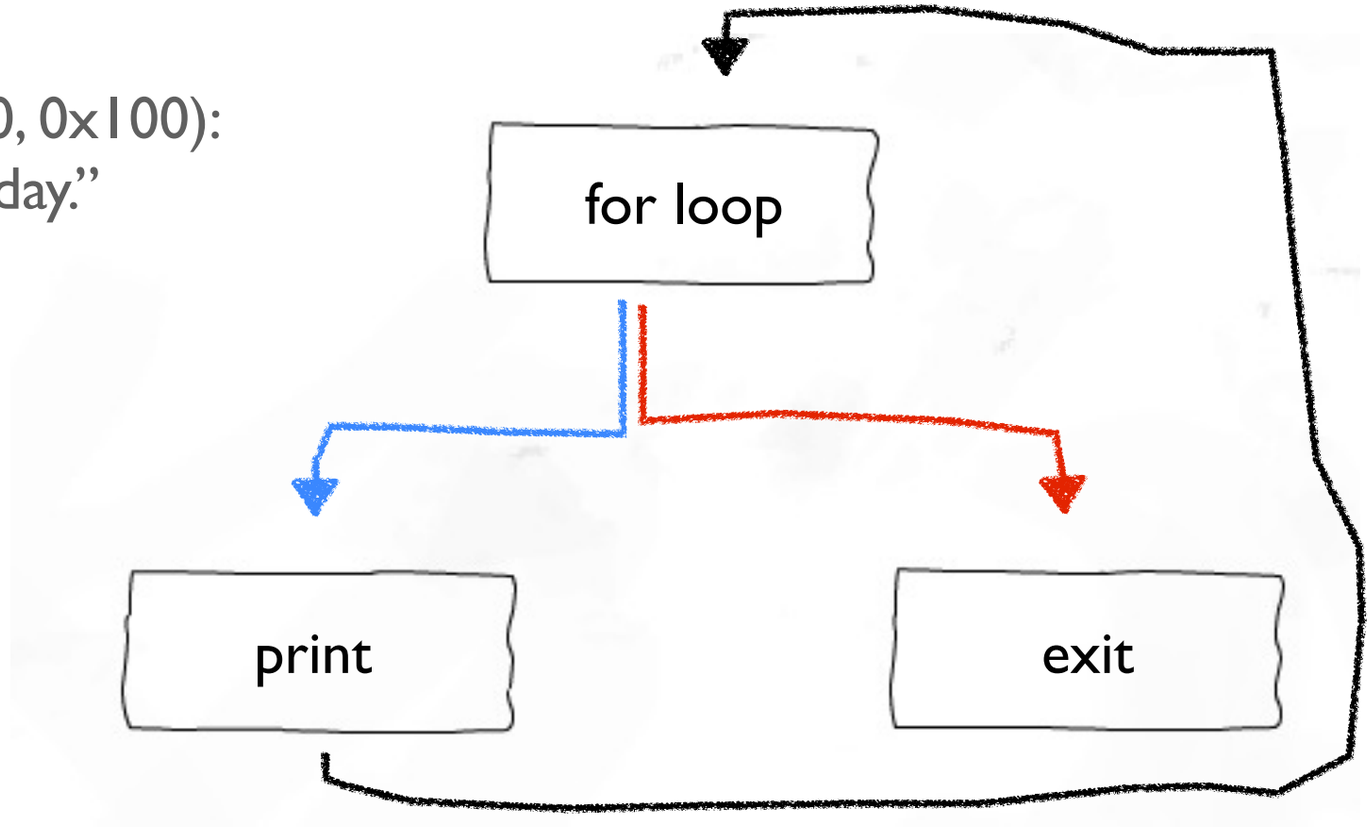
```
void func(char *str) {
    char buf[256];
    int x;
    for (x=0;x<=sizoef(buf);x++)
    {
        buf[x] = str[0];
    }
    .....
    .....
}
```

[vulnerable code]

Hunting interesting functions

- Loop detection is needed
- There are many loop types

```
for i in range(0, 0x100):
    print "%d 0day."
exit
```



Hunting interesting functions

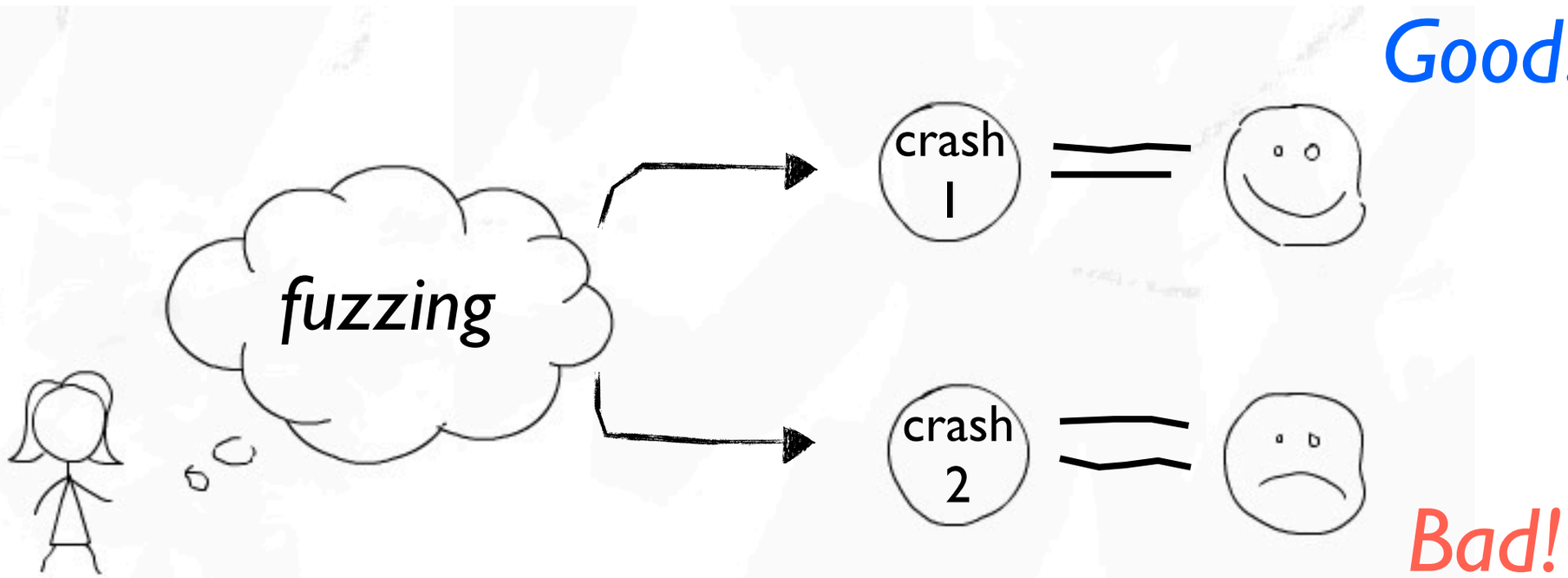
- Loop but ecx or esi is constant
- No interesting function calls (strcpy, ++)
- No integer operation mismatches (ja/jb vs jg/jl)
- Reference: Halvar's BH 2004, Edgar's Syscan 2012

!exploitable

- When you fuzz, you get MANY BUGS (crashes!)

!exploitable

- How do you figure out which one is exploitable but you have 10,000 crashes?



- Automation is needed

!exploitable

- MS provides crash analyzer "!exploitable"
- It says if a crash is exploitable or not
- It assumes an attacker can control these 3
  - destination address
  - move length
  - source address
  - like *memcpy(dst, src, length)* is under him

Be imaginative

- No limit to fuzzing target
- SYSCAL/SMS/Baseband/SCADA(PLC)
- URL/VM-Communication/Filesystem
- CPU instruction
  - recent instructions are complex

Even you know all those issues

- You may think you're very behind
- *Since 2008, SAGE has been running 24/7 on an average of 100-plus machines/ cores automatically fuzzing hundreds of applications in Microsoft security testing labs. This is more than 300 machine- years and the largest computational usage ever for any SMT (Satisfiability Modulo Theories) solver, with more than 1 billion constraints processed to date.*
- And there are definitely more groups who do advanced fuzzing

Tips for beginners

Setting your basement

- OS: Windows XP first!
  - Easy to debugging
  - Almost every RE tool works on XP
  - (I use linux for developing tools)
- Find bugs and debug/exploit them upon XP
- And port it for other versions of OS (win7/8)

Setting your basement

- VMWARE (or other VM) is mandatory
- Use snapshots
- Your OS will be messed up by your fuzzer
- Rebooting doesn't work time to time
- The migration feature is very gold

Language

- Don’t listen to others
  - how many friends of you have given you a tip for learning english? how many tips worked for you?
- Just choose one whatever you like
- But if you still need a recommendation, python is my answer
- Many hack libraries are written in python
- Ruby is getting popular

Language

- If you use C for fuzzing because you just want to feel you're l33t, you're wrong
- Realize what is your goal
- (No offense, C is my favorite language and I know many good hackers doing fuzzing with C because they love C.)

RE tools around you

- Do not invent wheels again
- A bunch of tools out there which you can use for making a homebrew fuzzer
- Example)
  radamsm + little python script = win

But do not believe tools too much

- Unfortunately, some RE tools are naive
- You may spend weeks to fix tools' bugs
- You may be end up with developing your own scripts / engines

Figure out what features you can add

- Great tools out there
- Pitch, Sully, and etc
- But you never get satisfied with them
- Find what you can make it better

Spend your time right

- Imagine we have 2 problems
- One thing very challenge but you may improve only 2% better performance
- Other one very easy and you may improve 20% better performance
- Also, I'd buy a new computer rather than making 2% better performance spending 50 hours

Hack like a champion

- Your fuzzer might be very naive and silly
- But not many people really do fuzzing
- Feel confident! Hack like a champ!

Code coverage is most important

- Try to find a way to do better coverage
- Even you spend 10000 hour fuzzing, it doesn't say you're doing right
- Always check if you're doing right
- For example: HWP takes compressed files in default which means you're not fuzzing HWP format unless you decompress it

Understand the principles first!

- You may get exceptions (crashes) but you're doing wrong if you have no ideas about them
- GO READ "The art of security software assessment" if you've not read!
- Or you only look for strcpy() FOREVER

Mom, I hate quiz

Figure out what is wrong with this code

snprintf trick

#1

```
int num;
char buf[256];

num = snprintf(buf, sizeof(buf), argv[1]);

if(num >= sizeof(buf) - 2) {
   printf("I hate you\n");
   exit(0);
}
```

snprintf trick

#1

```
int num;
char buf[256];

num = snprintf(buf, sizeof(buf), argv[1]);

if(num >= sizeof(buf) - 2) {
   printf("I hate you\n");
   exit(0);
}
```

- OS dependency, works differently
- Windows against Linux
- Return value and null-termination

GetFullPathName bad #2

```
if(GetFullPathName(c_file, length, buf, &o) ==0) {
    printf(“bad\n”);
    exit(0);
}
```

GetFullPathName bad #2

```
if(GetFullPathName(c_file, length, buf, &o) ==0) {
    printf(“bad\n”);
    exit(0);
}
```

- API understand (Return value)
- if buf is not enough to contain,
  “RET” is the number required to do

GetFullPathName bad #2

If the lpBuffer buffer is too small to contain the path, the return value is the size, in TCHARs, of the buffer that is required to hold the path and the terminating null character.

[MSDN]

- And the API doesn't touch the lpBuffer
- *Uninitialized* is a very good friend *sometimes*

wcsncpy madness #3

```
wchar_t buf[256];

wcsncpy(buf, user_controlled_buf, sizeof(buf));
```

wcsncpy madness #3

```
wchar_t buf[256];

wcsncpy(buf, user_controlled_buf, sizeof(buf));
```

- API understand (Arguments)
- The 3rd argument is number in wide characters

Mom, I told you
I hate quiz!!!!

Auditing code

- Auditing code makes you a better fuzzing programmer
- Find nice ideas while auditing code (or reversing the target)
- And apply them to your fuzzer

Easy target first

- GRETECH (Gomplayer)
- ESTSOFT (Alzip, Alyac)
- HANGUL HWP

Next level

- FLASH
- PDF
- MS WORD

Next next level

- Apache, PHP, IIS, SSH of course
- And SMS, PIC, Baseband and etc

- It's not about fuzzing itself
- How to set up your fuzzer on them?
- Even how do you code upon the environments?

ETC

- Fun is first, you don't have to be a pro like Dr. Charlie Miller
- Figure out values of your bugs
- Your *e*asy** bug might be my *e*asy** bug

Extra tips (Last word)

- Have fun with hunting attack vectors
- Go deep
- Contact people who do actual research
  - co-work is fantastic

Q/A

- Thanks to codeengn and certlab
- Passket will buy beer at after party
- Drink free beer
- SECUINSIDE will be on 11th July