Hijacking client software for fuzz and profit

how i met your pointer
ALTERNATIVE TITLES I CONSIDERED

- "I just met you and this is crazy but here's my pointer so jump to it maybe" (too long)
- "All your pointers are belong to..." (meh)
- "How I fuzzed your mother" (FAIL)

- Other names that could lead to discussion
Introduction.

Fuzzing 101. *yawn*

The need for a different approach.
  - Abusing the client.

A possible implementation. Boyka.

Experiment.

Conclusion.
$ whoami

The wiki page for "Physics Major" redirects to "Engineer."

Physics Major

$ whoami

True story, bro!
WHAT THIS IS ABOUT

- Interesting approach to software testing
- Touching things you are not supposed to
- Breaking stuff (if you're lucky!)
- Multiple references to pop culture
  - and chocolate!

Look for the ChocoQuiz Icon
What this is NOT about

- "Click & Hack" tool
  - There is juicy code though

- Although I find this pretty AWESOME
  - You may feel different about it
  - I did this in my spare time
  - You still have time to leave the room :)
  - Remember the chocolate
For educational purposes only...
Fuzzing 101
BITCH
PLEASE
Fuzzing is like violence: if it doesn‘t solve your problems, you are not using enough of it.
BRUTE FORCE
from sulley import *
from requests import ftp  # this is our ftp.py file

sess = sessions.session(session_filename="audits/freefloatftp.session")
target = sessions.target("192.168.1.11", 21)
target.netmon = pedrpc.client("192.168.1.11", 26001)  # NetMonitor (packets)
target.procmon = pedrpc.client("192.168.1.11", 26002)  # ProcMonitor (crashes)
target.procmon_options = { "proc_name" : "FTPServer.exe" }

sess.add_target(target)

sess.connect(s_get("user"))
sess.connect(s_get("user"), s_get("pass"))

sess.connect(s_get("pass"), s_get("cwd"))
sess.connect(s_get("pass"), s_get("dele"))
sess.connect(s_get("pass"), s_get("mdtm"))
sess.connect(s_get("pass"), s_get("mklc"))

sess.fuzz()
And Now

I Wait
CRASH! BOOM! BANG! HAHA!
Unable to disassemble at 20202020 from thread 472 caused access violation when attempting to read from 0x20202020

CONTEXT DUMP

EIP: 20202020 Unable to disassemble at 20202020
EAX: 00000216 ( 534) -> N/A
EBX: 00000002 ( 2) -> N/A
ECX: 0014d3c0 (1364928) -> F unt authority\systemzA (heap)
EDX: 7c90e514 (2089870612) -> N/A
EDI: 003b19d5 (3873237) -> (heap)
ESI: 0040a44e (4236366) -> N/A
EBP: 003b1298 (3871384) -> N/A
ESP: 00b2fc2c (11729964) ->
+00: 20202020 (538976288) -> N/A
+04: 20202020 (538976288) -> N/A
+08: 20202020 (538976288) -> N/A
+0c: 20202020 (538976288) -> N/A
+10: 20202020 (538976288) -> N/A
+14: 20202020 (538976288) -> N/A

disasm around:
0x20202020 Unable to disassemble

SEH unwind:
fffffffff -> kernel32.dll:7c839ad8 push ebp
There’s always a but

That’s all very nice.

But what if I don’t know the protocol?
There is NO documentation at all. :(((((
I CAN ALWAYS TRY DUMB FUZZING!
Think about checksums...

Checksum = SHA1(Data)
SHA1: 160 bits
P(right) = \(1/2^{160}\) \(\approx 1/10^{48}\)
\(10^{48} = 1k \cdot 1T \cdot 1T \cdot 1T \cdot 1T \cdot 1T\)

Dumb Fuzzing...
IS EVERYTHING LOST?

FAIL HARDER

THINK WRONG
The need for a *different* approach
WE ARE SECURE.
NOBODY KNOWS OUR PROTOCOL.

THE CLIENT DOES...
Simple Arguments

```assembly
; int __stdcall cgp_ArithmeticSender01(char *buf, int len, int)
cgp_ArithmeticSender01 proc near

var_FE = word ptr -0FEh
var.FC= qword ptr -0FCh
Dest= dword ptr -0F4h
var_4= dword ptr -4
buf= dword ptr 4
len= dword ptr 8
arg_8= dword ptr 0Ch

sub esp, 100h
mov eax, stackCookie
xor eax, esp
mov [esp+100h+var_4], eax
push ebp
mov ebp, [esp+104h+buf]
```

It gets exciting.
Detours
= userland hooking
= amazing stuff

= dynamic binary instrumentation
= AWESOME stuff !!!
Microsoft Detours

- Library for intercepting arbitrary Win32 binary functions.
- Interception code is applied dynamically at runtime.
- Replaces the first few instructions of the target function with an unconditional jump to the detour function.
- Replace or extend the target function.
Intel Pin

- Executable instrumented before running
  - Delay noticeable
  - Finds new code at runtime !!!
  - Packed/protected code is not a problem
  - Nor is Antidebugging :)
What can possibly go wrong?

Developer: “Can’t touch this!”
What can possibly go wrong?

"Hmm... Can't touch this?"
What can possibly go wrong?

"Actually, I can!"
LONG STORY SHORT...

Dance, Puppet!

Dance!
PLUMBING TIME
OVERVIEW (FROM A MILLION MILES AWAY)

Server

- Debugger
- Server software
- Communications module

Client

- Client software
- Debugger
- Communications module

Protocol

Event info.
Feedback
**OVERVIEW (FROM A THOUSAND MILES AWAY)**

Client software

- **f1**
- **f2**
- **f3**
- **f4**
- **f5**
- **f6**

**hooking**

- send()

(*) This step works like so... so... right now

Save state

Restore state
... SOMETHING WRONG MAY HAPPEN
NOT SURE IF THIS MAKES SENSE

OR I'M STARTING TO BELIEVE MY OWN BULLSHIT
I can "inject" some data into the server
By hijacking client execution at certain points
...
... aha...

Which. Points. Do. I. Use. ?!?!?!?
Anyone getting dizzy?
BRACE YOURSELVES, SHAMELESS AUTOPROMOTION IS COMING.
M*LF & PIN TRACER
Some cool features

- Mark dangerous functions
- Find immediate compares
- Mark switches
- Show paths between functions
- Find File IO
- Find Network IO
- Find Allocations
- Find dangerous "size params"
- Create IDA (connection) graphs
- Create "custom viewers"
- etc.
<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Block Address</th>
<th># Instr</th>
<th>Arithmetic/Logic Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x433c80</td>
<td>CRC32</td>
<td>0x433f20</td>
<td>8</td>
<td>75.00</td>
</tr>
<tr>
<td>0x433c80</td>
<td>CRC32</td>
<td>0x433cc0</td>
<td>116</td>
<td>71.55</td>
</tr>
<tr>
<td>0x433c80</td>
<td>CRC32</td>
<td>0x433ed0</td>
<td>19</td>
<td>68.42</td>
</tr>
<tr>
<td>0x428791</td>
<td>XorChainEncrypt</td>
<td>0x4287a5</td>
<td>6</td>
<td>50.00</td>
</tr>
<tr>
<td>0x428bb</td>
<td>Base64Decode</td>
<td>0x428767</td>
<td>9</td>
<td>33.33</td>
</tr>
<tr>
<td>0x428520</td>
<td>Rc4</td>
<td>0x42854b</td>
<td>19</td>
<td>36.84</td>
</tr>
<tr>
<td>0x427b37</td>
<td>DecryptString</td>
<td>0x427b4d</td>
<td>12</td>
<td>33.33</td>
</tr>
<tr>
<td>0x427b01</td>
<td>StringEncrypt</td>
<td>0x427b16</td>
<td>9</td>
<td>33.33</td>
</tr>
<tr>
<td>0x42633c</td>
<td>MersenneTwister</td>
<td>0x4263a2</td>
<td>14</td>
<td>57.14</td>
</tr>
<tr>
<td>0x42633c</td>
<td>MersenneTwister</td>
<td>0x426355</td>
<td>14</td>
<td>57.14</td>
</tr>
<tr>
<td>0x426307</td>
<td>MersenneTwist...</td>
<td>0x426315</td>
<td>11</td>
<td>54.55</td>
</tr>
</tbody>
</table>

28 blocks from a total of 13292 blocks matched with the above settings.

http://pnx-tf.blogspot.com/
Differential Debugging

- Hook every function -> log hits.
- 1st run. Exercise as many functionality as possible.
- 2nd run. Focus on the interesting feature.
- Compare both -> filter out.

<table>
<thead>
<tr>
<th>Function_1</th>
<th>Function_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI_stuff</td>
<td>GUI_stuff</td>
</tr>
<tr>
<td>Windows_stuff</td>
<td>Windows_stuff</td>
</tr>
<tr>
<td>Function_2</td>
<td>Login_stuff</td>
</tr>
<tr>
<td>Thread_sync</td>
<td>Thread_sync</td>
</tr>
<tr>
<td>Function_3</td>
<td>Encryption_stuff</td>
</tr>
<tr>
<td>[...]</td>
<td>[...]</td>
</tr>
</tbody>
</table>
Differential Debugging

- Hook every function -> log hits.
- 1st run. Exercise as many functionality as possible.
- 2nd run. Focus on the interesting feature.
- Compare both -> filter out.
BUILD YOUR WEAPON
Epic ass kicking

WAKE UP!
You're gonna miss the good stuff!!!
Finding possible weak spots
Finding possible weak spots

; Attributes: bp-based frame

; cdecl assemblePacket(char *login, char *szPacket, unsigned int len)

assemblePacket proc near

var_F8= byte ptr -0F8h
szSecretString= byte ptr -34h
szPKlen= byte ptr -20h
delimiter= dword ptr -0Ch
var_4= dword ptr -4h
login= dword ptr 8
szPacket= dword ptr 0Ch
len= dword ptr 10h

push ebp
mov ebp, esp
sub esp, 0F8h
push ebx
push esi
push edi
lea edi, [ebp+var_F8]
mov ecx, 3Eh
mov eax, 0CCCCCCCC
rep stosd
mov eax, __security_cookie
xor eax, ebp
mov [ebp+var_4], eax
mov [ebp+delimiter], offset a_ "\n"
mov eax, dword ptr ds:aBruconrocks;
"BruCONrocks"
mov dword ptr [ebp+szSecretString], eax
mov ecx, dword ptr ds:aBruconrocks+4
mov dword ptr [ebp+szSecretString+4], ecx
mov edx, dword ptr ds:aBruconrocks+8
mov dword ptr [ebp+szSecretString+8], edx
lea eax, [ebp+szSecretString]
push eax ; src
CHEATING...

- Calculate login length
- Append the length (ASCII) to the login string.
- Append a “custom“ string
- *Encrypts* everything

Server: Length value used to `malloc()` & `strcpy()`
STAND BACK

I'M GOING TO TRY SCIENCE
WHERE TO GO FROM HERE

- Better static / dynamic analysis
  - Automatization
  - Heuristic based
- Save / restore snapshot
  - Full emulation (Thx @pleed_ !)
  - Qemu-dbi?
You can lurz at my code at:

https://github.com/carlosgprado/Boyka

@m0n0sapiens
carlos.g.prado@gmail.com
IT'S BEEN LOVELY BUT I HAVE TO SCREAM NOW