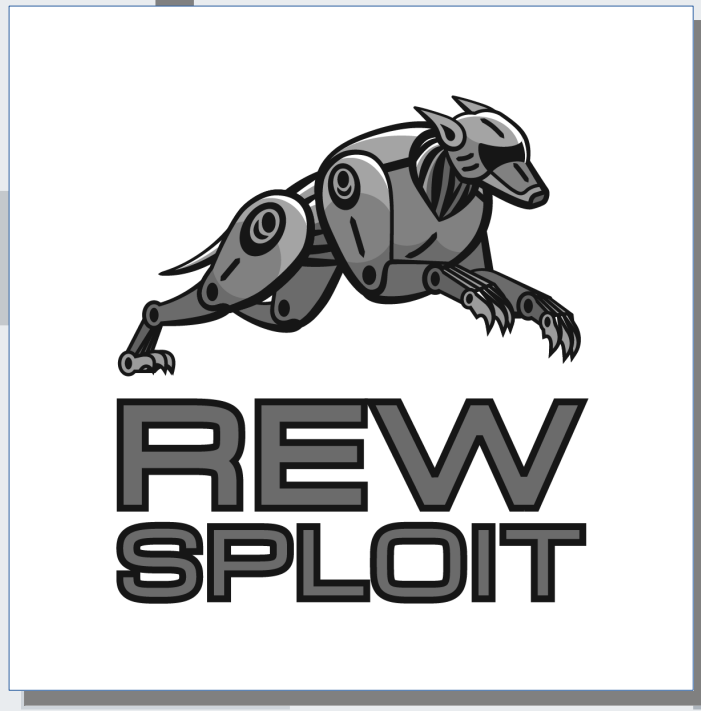


REW-sploit

dissect payloads with ease

Insomni'hack 2022
Cesare Pizzi @red5heep

About me:

Security guy at 

I like Reverse Engineering a lot, on both hardware and software.

Most of my more recent work is on github at https://github.com/cecio/

I like to participate to OS development and I contribute to security projects (like Volatility, OpenCanary, Cetus).

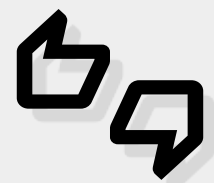
Also, I have some personal projects, like SYNwall and...the one I'm going to present today.

Random thought #01

We have a lot of “standard” tools out there to help red-teamer in doing their daily job. Metasploit and CobalsStrike are the most famous. But, we miss a blue-team generic tool, something that can help us in analyzing these payloads

Random thought #02

Can we create an equivalent tool for the Blue-Teamer, in order to help him? I need something like that and I want to make it OpenSource



Random thought #03

It will be a mess, I’ll regret this, but let’s try...

Standing on the shoulder of giants

I love Open Source Software and I like to use it when it's possible. I didn't do everything by myself, but I based the work on two well known tools/frameworks:

Unicorn Engine:

https://www.unicorn-engine.org/

Speakeasy Emulator (based on Unicorn as well):

https://github.com/mandiant/speakeasy

The Approach 1/3

- The initial focus of the tool was analyse Windows Metasploit x86/64 bits payloads.

- Right now I'm trying to evolve it in something more generic adding several support layers

- Support to other tools has been added (CobaltStrike, Donuts, etc). This is done at two leveles: REW-sploit itself or to the underlying tools (Unicorn and SpeakEasy, usually with pull-requests)

The Approach 2/3

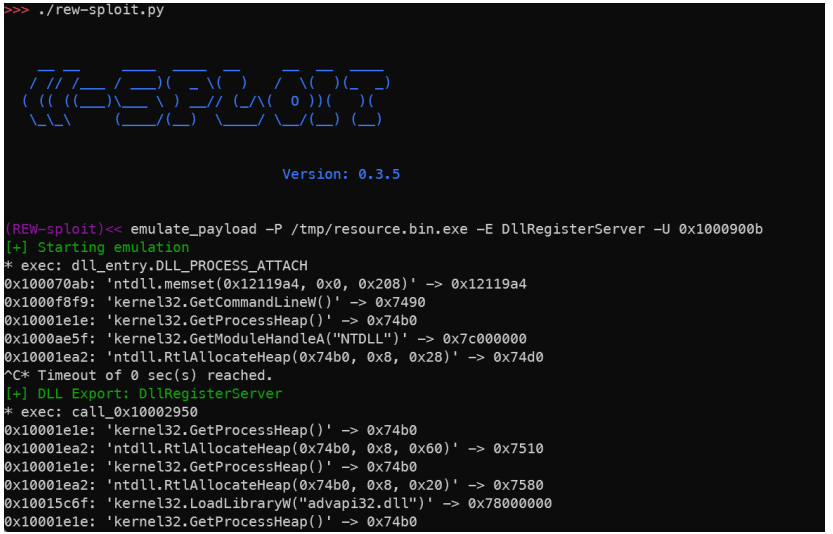
- The basic idea is this: the underlying tools may not work out-of-the-box with all the payloads, several customization are needed. I apply this customization together with some additional checks I may find useful

- Basic operations are applied on binary files (shellcode, EXE and DLL) and possibly on PCAP files if available

The Approach 3/3

- The interface is a typical CLI where you can interact with the OS, by running output manipulation commands (like “grep”), through a “cmd2” interface

- REW-sploit can automatically detect the payload format (EXE, DLL or shellcode) and then start the emulation of the code. Depending on the input, it can help you in deobfuscating the code, extract API calls, keys and artifacts.



Advantages

Static Tools can easily be fooled by obfuscation, encryption and other tricks. Code Emulation can overcome this.

By emulating the code, you can interact with it: that means that you can change the flow of the program depending on what you want to do.

Combined usage with other RE tools (like debuggers or disassembler) can speed up the RE process.

Disadvantages

Emulation can be a fragile process: a lot of thing should be considered and sometimes it just breaks!

Emulation can be slow in certain situations.

It’s a new tool: learning and practice is required to get the most out of it.

REW-sploit emulation features

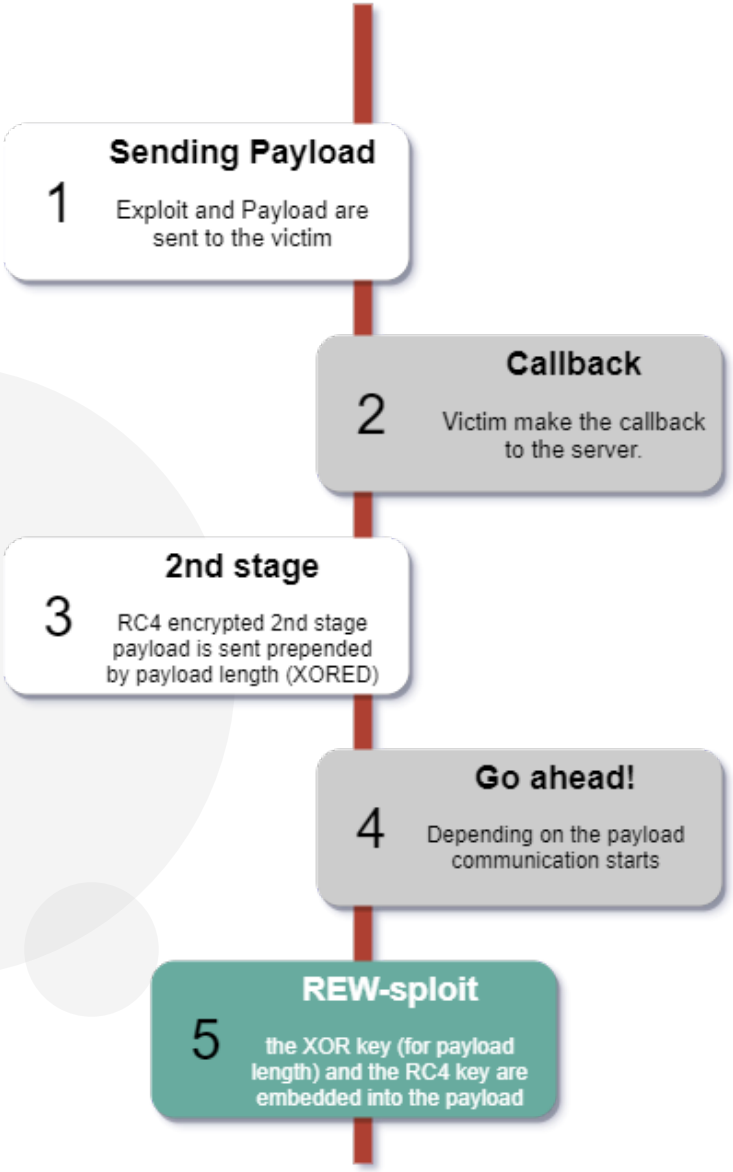
- The tool is able to automatically detect MSF and CobaltStrike payloads and manage them
- Best results are seen with PIE binaries, even if EXE and DLL can be emulated as well. Remeber: this is not a sandbox, it’s an emulation platform.
- The tool can emulate any executable, but it may breaks. Sometimes is an easy fix, sometimes not. But you can open an Issue in Github and we can take a look together ;-)

Metaspoloit Payloads

MSF Payloads may fall under several groups:

- RC4 encrypted callback code
- a less common “chacha” encrypted shell
- encrypted Meterpreter shell
- many other...

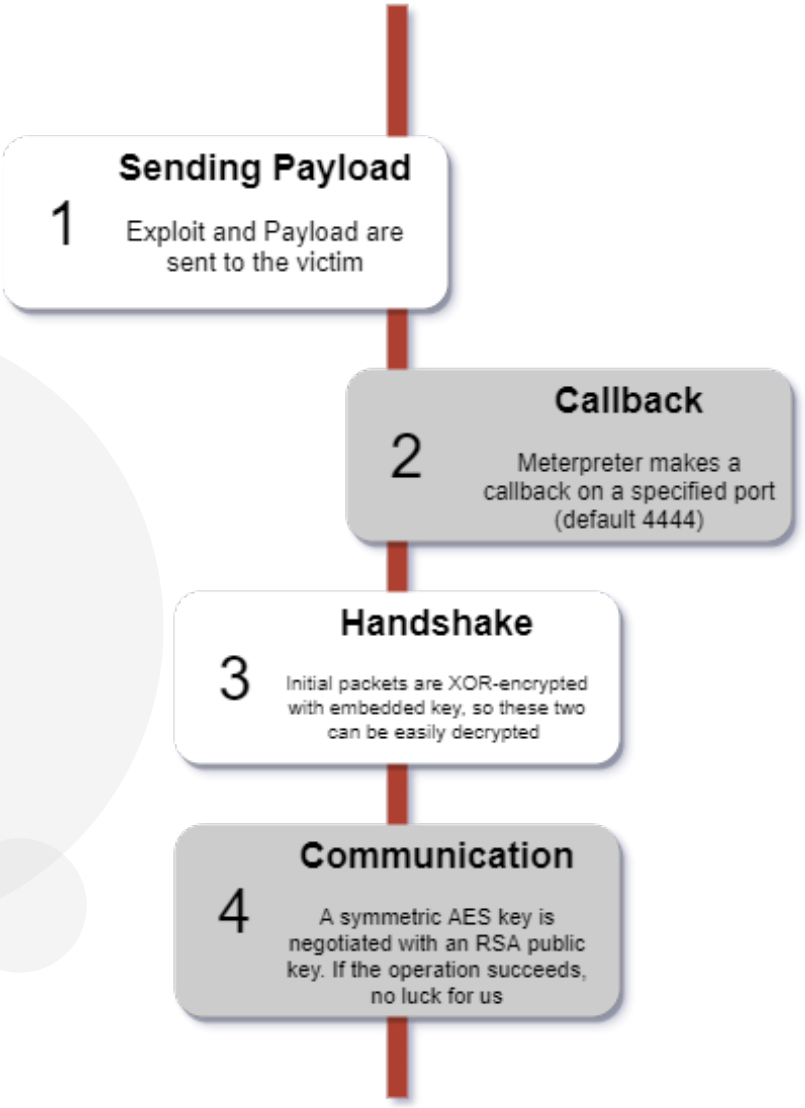
RC4 Encrypted Payloads



DEMO

RC4 and 2nd stage extraction

Meterpreter encrypted shell 1/3



Meterpreter encrypted shell 2/3

We still have a possibility: if the RSA import fails on the victim, Meterpreter silently fall back to something easier to manage: the symmetric AES key is sent to Meterpreter server just xored and REW-sploit can read it

Meterpreter encrypted shell 3/3

If we want to monitor attacker activities, we can force this in a couple of ways:

- By following the POC||GTFO article "Exploiting weak shellcode hashes" (Meyers-Sultanik)
- By patching the payload (**FF 15 58 10 02 10 85 C0 *74* AC**) changing **74** (JE) in **75** (JNE)

DEMO

Meterpreter decryption

CobaltStrike beacons 1/3

The support for CobaltSrike is present and working, even if it needs improvement: right now beacons are automatically recognized by REW-sploit.

In addition, if CobaltStrikeParser (from SentinelOne) is installed on the system, the configuration will be dumped as well

CobaltStrike beacons 2/3

As you know CS beacons can be heavily customized, so configuration extractor may not work as expected. Anyway, the emulation can overcome to this, by simply following the code:

Example on EMOTET Epoch 5 sample (thanks to https://malware-traffic-analysis.net)

```
0x140004c08: 'kernel32.VirtualAlloc(0x0, 0x400, 0x3000, "PAGE_EXECUTE_READWRITE")' -> 0x50000
0x500ef: 'kernel32.LoadLibraryA("wininet")' -> 0x7bc00000
0x50107: 'wininet.InternetOpenA(0x0, 0x0, 0x0, 0x0, 0x0)' -> 0x20
0x50129: 'wininet.InternetConnectA(0x20, "[redacted]e.com", 0x1bb, 0x0, 0x0, 0x3, 0x0, 0x0)' -> 0x24
0x50148: 'wininet.HttpOpenRequestA(0x24, 0x0, "/wp-includes/links.png", 0x0, 0x0, 0x0, "INTERNET_FLAG_DONT_CACHE |
INTERNET_FLAG_IGNORE_CERT_CN_INVALID | INTERNET_FLAG_IGNORE_CERT_DATE_INVALID | INTERNET_FLAG_KEEP_CONNECTION | INTERNET_FLAG_NO_UI |
INTERNET_FLAG_RELOAD | INTERNET_FLAG_SECURE", 0x0)' -> 0x28
0x50172: 'wininet.InternetSetOptionA(0x28, 0x1f, 0x1211e20, 0x4)' -> 0x1
0x5018c: 'wininet.HttpSendRequestA(0x28, "Host: vk.com\r\\nConnection: close\r\\nAccept-Encoding: gzip\r\\nUser-Agent: Mozilla/5.0 (
Windows Phone 10.0; Android 6.0.1; Microsoft; RM-1152) AppleWebKit/537.36 (KHTML, like Gecko)\r\\n", 0xffffffffffffffff, 0x0, 0x501f9)' ->
0x1
0x5034d: 'kernel32.VirtualAlloc(0x0, 0x400000, 0x1000, "PAGE_EXECUTE_READWRITE")' -> 0x450000
0x5036b: 'wininet.InternetReadFile(0x28, 0x450000, 0x2000, 0x1211da0)' -> 0x1
0x5036b: 'wininet.InternetReadFile(0x28, 0x451000, 0x2000, 0x1211da0)' -> 0x1
0x450012: Unhandled interrupt: intnum=0x3
0x450012: module_entry: Caught error: unhandled_interrupt
```

CobaltStrike beacons 3/3

Customization to the beacons (like for example custom sleep-mask), will be automatically emulated and decoded, so you can inspect the initial beacons actions

DEMO

CobaltStrike dumping and emulation

How REW-sploit enhance emulation: Fixups

Sometimes, especially with heavily obfuscated and self modifying code, emulation breaks.

REW-sploit implements some manual fixups to be able to complete the emulation in most of the case.

Fixups must be manually enabled.

Fixups #1

```
#
# Fixup #1
# Unicorn issue #1092 (XOR instruction executed twice)
# https://github.com/unicorn-engine/unicorn/issues/1092
#               #820 (Incorrect memory view after running self-modifying code)
# https://github.com/unicorn-engine/unicorn/issues/820
# Issue: self modfying code in the same Translated Block (16 bytes?)
# Yes, I know...this is a huge kludge... :-/
#
```

Fixups #2

```
#
# Fixup #2
# The "fpu" related instructions (FPU/FNSTENV), used to recover EIP, sometimes
# returns the wrong addresses.
# In this case, I need to track the first FPU instruction and then place
# its address in STACK when FNSTENV is called
#
```

Fixups #3

```
#
# Fixup #4
# Stack too small (not enough values stored)
#
# Some obfuscator/evasion technique try to access some values on the stack
# (like for example SGN https://github.com/EgeBalci/sgn.git):
#
#     cmovne ax, word ptr [esp + 0xfa]
#
# In this case the emulation fails with an "invalid_read" since ESP is too
# close to the top of the stack. This creates some 'fake' values.
#
```

Additional Support
Donut 1/2

Donut is a well-known utility to create Position Independent Code out of EXE, DLL and other executables.

REW-sploit has some internal rules to identify the Donut “stub” and act accordingly in the emulation.

Additional Support
Donut 2/2

Donut uses a API exports enumeration based on hashes as many PIC do. This is very CPU intensive. In this case REW-sploit implements a sort of shortcut to unhook some of the slowest parts of emulation when a Donut stub is detected.

Example of Donut Fixup

```
def hook_mapviewofsection(emu, api_name, func, params):
    """
    Hook for ZwMapViewOfSection
    This one is needed for Donut EXE emulation.
    The very first call of this API is done with protection
    PAGE_READWRITE (0x04). After a while unicorn drops an read access
    violation. This does not happen if I patch the protection to
    PAGE_EXECUTE_READWRITE (0x40)
    """
```

Additional Support PEzor/SGN

PEzor is another very interesting PE packer with several functionalities (one of them the use of EgeBalci/SGN implementation).

Specific Fixups has been added to emulate the resulting code.

Antidebug

New feature in version 0.4 (just released): a new command to identify anti-debug techniques used in executables/shellcode.

There are tons of antidebug tricks, I'll add more in the future, so far we have (some examples):

Antidebug

[#] Direct access to PEB!BeingDebugged at 0x415a6a

[#] Direct access to PEB!NtGlobalFlag at 0x415a9e

Antidebug

```
[#] Call to QueryPerformanceCounter() at 0x140014367
[#] IsDebuggerPresent() at 0x140015dad
[#] CheckRemoteDebuggerPresent() at 0x140015ddc
[#] Suspect NtQueryInformationProcess() at 0x140015e5a
[#] Suspect NtQueryInformationProcess() at 0x140015ee7
[#] Suspect NtQuerySystemInformation() at 0x140015fd1
[#] Direct access to PEB!BeingDebugged at 0x14001601c
[#] Direct access to PEB!NtGlobalFlag at 0x140016053
[#] Suspect access to HeapBase (may be used to access Flags and ForceFlags) at 0x1400160bb
[#] GetProcAddress() of CRSS.EXE at 0x1400160f4
[#] Exclusive CreateFileA() on current process at 0x140016146
[#] Call to GetLocalTime() at 0x14001615f
[#] Call to GetSystemTime() at 0x140016185
[#] Call to GetTickCount() at 0x1400161a4
[#] Call to QueryPerformanceCounter() at 0x1400161cf
[#] Call to timeGetTime() at 0x1400161ee
[#] Call to VirtualProtect() on "Return Address" at 0x140011e9b
```

DEMO

Antidebug

How to customize 1/2

emulate_rules.py

```
#
# Payload Name: [MSF] windows/meterpreter/reverse_tcp_rc4
# Search for  : xor esi,0x<const>
# Used for    : this xor instruction contains the constant used to
#               encrypt the lenght of the payload that will be sent as 2nd
#               stage
# Architecture: x32/x64
#
yara_reverse_tcp_rc4_xor = 'rule reverse_tcp_rc4_xor {                \
                            strings:                                  \
                                $opcodes_1 = { 81 f6 ?? ?? ?? ?? }    \
                            condition:                                \
                                $opcodes_1 }'
```

How to customize 2/2

emulate_payload.py

```
#####################################
# YARA RULES MATCHING SECTION START #
#####################################

# Look for "xor esi,0x<const>"
if rule_reverse_tcp_rc4_xor_32.match(data=opcodes_data):

    try:
        # Replace the value to read just 8 bytes
        xorconst = opcodes & 0xFFFFFFFF
        xorval = struct.unpack("<I", struct.pack(">I", xorconst))[0] ^ 8
        emu.reg_write(e_arch.X86_REG_ESI, xorval)
        cmd2.poutput(Fore.MAGENTA + '[+] XOR constant for payload length: %s - 0x%x' % (struct.pack("<I", xorconst),
                                                                                        struct.unpack("<I", struct.pack(">I",
                                                                                        xorconst))[0]) +
                     Style.RESET_ALL)
```

Performances 1/2

Performances are a concern: emulation slow down things, no way around.

The additional fixups and harnesses created also may heavily impact the execution time.

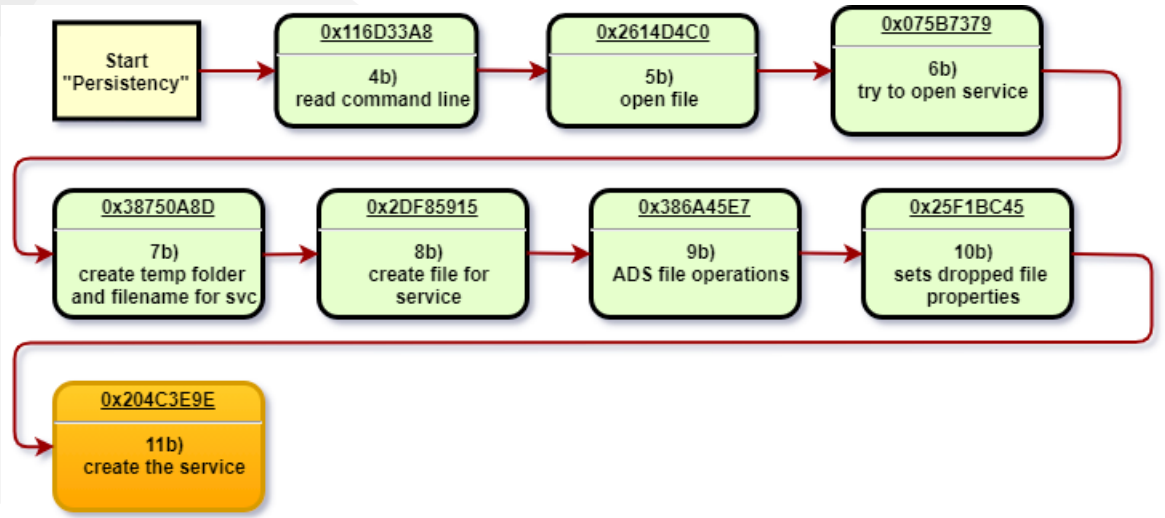
A quick workaround has been implemented, at least for a quick analysis.

Performances 2/2

It is possible, with the proper option, to disable all the code hooks done by REW-sploit, in order to speed up emulation.

The option allows also specify a memory address to re-enable the hooks, to start from a specific execution point.

Use case #1: execution interaction



https://github.com/cecio/EMOTET-2020-Reversing

Use case #2:
Dumping Thread

(REW-sploit)<< emulate_payload -P /tmp/lync.exe -T

...

[+] Dumping CreateThread (complete dump saved in /tmp/tmpz2lhgw6z/0x468840.bin)

...

DEMO

Dumping Thread

Use case #3:
Dumping Allocation

(REW-sploit)<< emulate_payload -P /tmp/lync.exe -M

...

[+] Dumping VirtualAlloc (complete dump saved in /tmp/tmpz2lhgw6z/0x468840.bin)

...

DEMO

Dumping Allocation

Use case #4: Dumping Files

(REW-sploit)<< emulate_payload -P /tmp/lync.exe -W

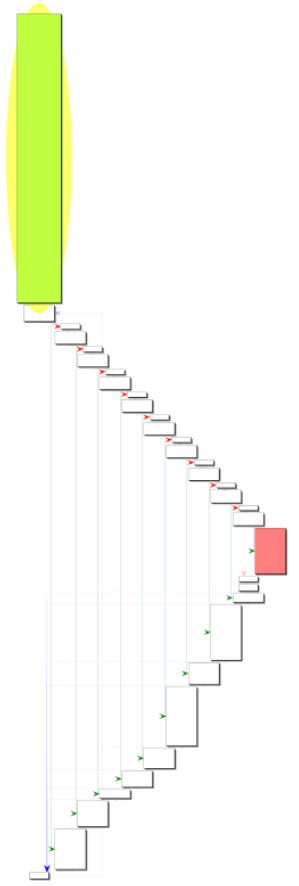
...

[+] Dumping WriteFile (complete dump saved in /tmp/tmpz2lhgw6z/0x468840.bin)

...

Use case #5: deobfuscation help

O-LLVM can make code pretty unreadable



DEMO

Deobfuscation

Caveats

- REW-sploit is not a sandbox
- Underlying tools sometimes need quick & dirty help:

Adding “obscure” API calls to msvcrt.py file, to emulate them.

“No idea what I’m doing here” mode on.

```
@apihook('__iob_func', argc=1, conv=e_arch.CALL_CONV_CDECL)
def __iob_func(self, emu, argv, ctx={}):

    return 0
@apihook('___lc_codepage_func', argc=0, conv=e_arch.CALL_CONV_CDECL)
def ___lc_codepage_func(self, emu, argv, ctx={}):

    return 0
```

The next steps

A lot of things need to be done:

- Improve emulation stability with new harnesses
- Improve performances
- Improve Anti-debug coverage
- ...

Please open issues on GitHub on what you think is more useful!

References

- **Unicorn**: https://www.unicorn-engine.org/
- **SpeakEasy**: https://github.com/mandiant/speakeasy
- **Cmd2**: https://github.com/python-cmd2/cmd2
- **SentinelOne CS Parser**: https://github.com/Sentinel-One/CobaltStrikeParser
- **Donut**: https://github.com/TheWover/donut
- **Pezor**: https://github.com/phra/PEzor
- **SGN**: https://github.com/EgeBalci/sgn
- **Malware Traffic Analysis**: https://www.malware-traffic-analysis.net

Thank you!

https://github.com/REW-sploit/REW-sploit

Cesare Pizzi @red5heep