black hat USA 2021

August 4-5, 2021

ARSENAL

REW SPLOIT

Cesare Pizzi (@red5heep)

About me:

Security guy at Sorint.lab.

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 try to contribute to projects (like Volatility, OpenCanary, Cetus).

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

REW-sploit

Random thought #1

There are some standard tools out there to help red-teamer (or may be malicious actors) to carry on attacks: Metasploit and CobaltStrike are the most famous in the field.
But if we are a blue-team member? We have a lot of scripts or tools, but not a real “one-stop tool station” for this.

Random thought #2

What if we can create a tool to help blue teams in the same way MSF and CS are helping red teams?

Standing on the shoulders of giants

Well, I have to admit: 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:

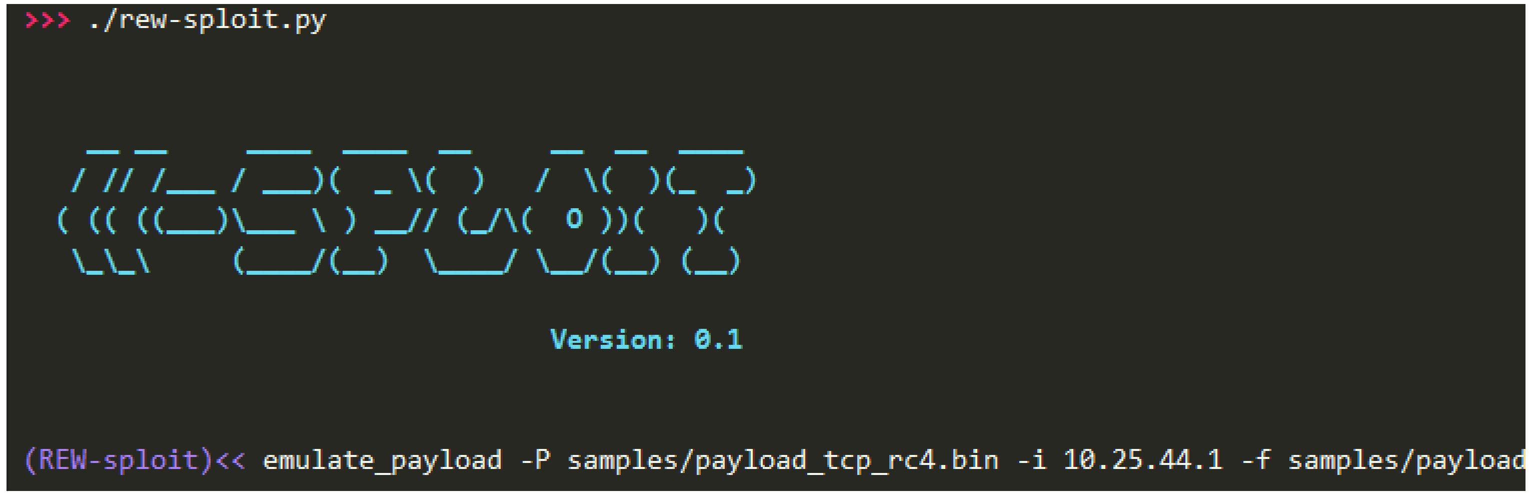
Unicorn Engine:
https://www.unicorn-engine.org/

Speakeasy Emulator (based on Unicorn as well):
https://github.com/fireeye/speakeasy

REW SPLOIT

General approach

- The tool is currently focused on Metasploit6 x86/x64 payloads/shells + a more generic approach for meterpreter shells.
- Also Cobalt-Strike is part of the game, even if in early stages.
- The interface is a CLI, for easy scripting, just in case ☺



```
>>> ./rew-sploit.py



    __ __     ____  ____  __     __  __  ____
   / // /___ / ___)(  _ \(  )   /  \(  )(_  _)
  ( (( ((___)\___ \ ) __// (_/\(  O ))(   )(
   \_\_\    (____/(__)  \____/ \__/(__) (__)

                              Version: 0.1


(REW-sploit)<< emulate_payload -P samples/payload_tcp_rc4.bin -i 10.25.44.1 -f samples/payload
```

General approach

- It creates some custom harnesses to the payloads, to emulate them and extract relevant information
- Basic operations are done on shellcode files and PCAPs. You can pass to REW-sploit a PCAP and an initial shellcode:

    REW-sploit will emulate shellcode

    Encryption keys will be extracted

    Parsing the PCAP it will identify the relevant connection

    It will decrypt the conversation and it will dump content for further analysis

REW-sploit features

Emulation main features are:

- Emulation will automatically detect the MSF payload or CS beacon
- Emulation will automatically detect if the specified file is an EXE, DLL or shellcode. In case of DLLs it will emulate all the exported functions
- You can pass a to the emulation files coming from other sources (so no MSF or CS), it may work or not, depending on your luck! But you can open an Issue and we can look into that.

Metasploit Payloads

MSF payloads/artifacts have may fall under several different groups. Some of them are:

- rc4 encrypted callback code
- chacha encrypted shell (MSF5, not sure how much used)
- encrypted meterpreter shell
- plus many other I'm working on...

Payloads details and how REW-sploit can help

RC4 encrypted payloads:

- standard flow:
  - → exploit + payload
  - ← payload makes callback
  - → RC4 encrypted 2nd stage payload is sent prepended by payload length (XORED)
  - ←→ depending on the 2nd stage payload additional connections are made
- the XOR key (for payload length) and the RC4 key are embedded into the payload
- by extracting them, REW-sploit is able to decrypt the payload sent

Demo!

Payloads details and how REW-sploit can help

CHACHA encrypted payloads:

- standard flow:
  - ➡ exploit + payload
  - ⬅ payload makes callback with UUID
  - ➡ CHACHA key and nonce are selected from MSF db and encrypted shell starts

- CHACHA key and nonce are embedded in payload
- by extracting them, REW-sploit is able to decrypt the conversations

Demo!

Payloads details and how REW-sploit can help

Meterpreter encrypted shell #1:

- This is a bit more articulated: meterpreter payloads makes a callback to MSF (default port 4444)
- The first two exchanged packets of the connections are XOR-encrypted with the key embedded in the packet, so these two can be easily decrypted
- All packets are TLV (Type-Length-Value) lists, with the different commands encoded
- With these two packets meterpreter try to negotiate a symmetric AES key crypted with an RSA public key. If the operation succeeds, no luck for us

Payloads details and how REW-sploit can help

Meterpreter encrypted shell #2:

- At this point, what REW-sploit can do is to automate the identification of this 2-way-handshake and identify a meterpreter connection (not so obvious if not done on port 4444), dumping RSA public key and encrypted symmetric key.
- **BUT**: if the RSA import fails on the victim, <u>meterpreter silently fall back </u>to something more manageable: <u>the symmetric AES key is sent to meterpreter server just xored</u> and REW-sploit can read it

Payloads details and how REW-sploit can help

Meterpreter encrypted shell #3:

we can force this fall-back in several way:

- 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)
- With this forced fallback in place, REW-sploit is now able to decode the whole conversation (IV for AES is embedded and XORED in each packet).

The complete set of Meterpreter commands is imported to be decoded in clear

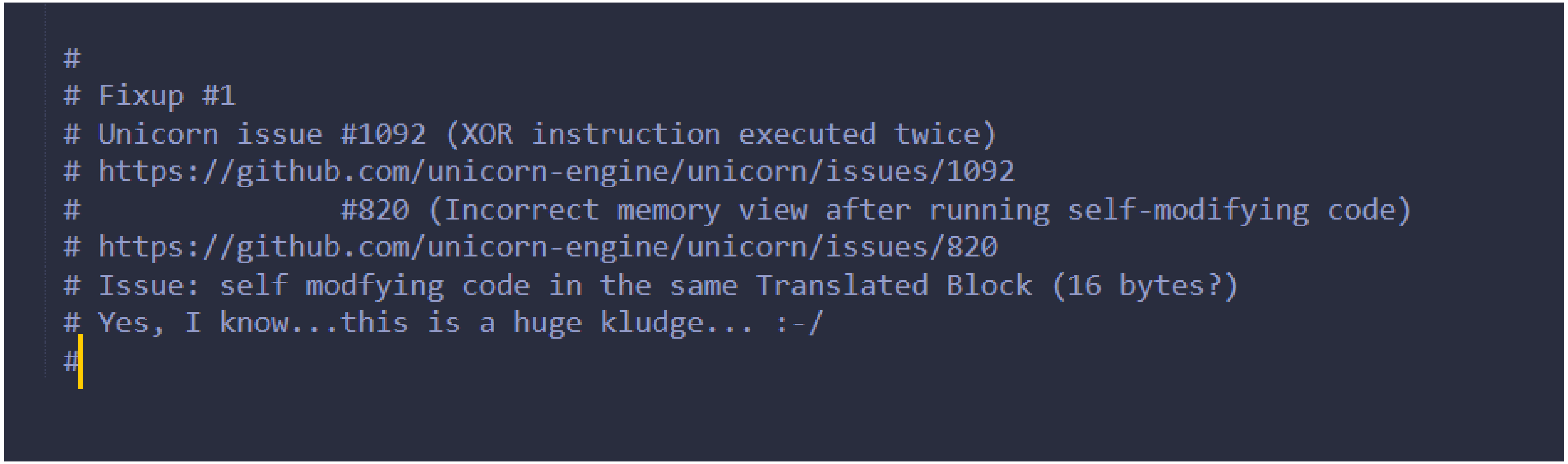
Demo!

Payloads encoding and fixups

With Metasploit you can choose several different encoders to hide your shellcode on target system (shikata-ga-nai and others):

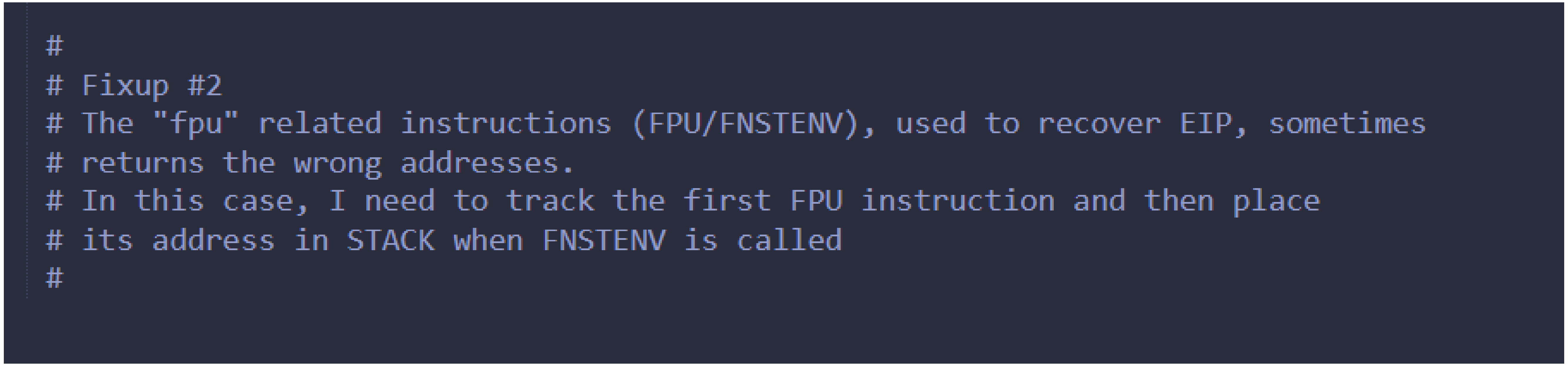
- The REW-sploit emulation (see approach section) is able to follow the entire execution and unroll the code
- In some cases the highly obfuscated code breaks the emulation, but REW-sploit implements some ad-hoc fixes to be able to complete the extraction successfully, by using some custom harnesses

Fixups examples



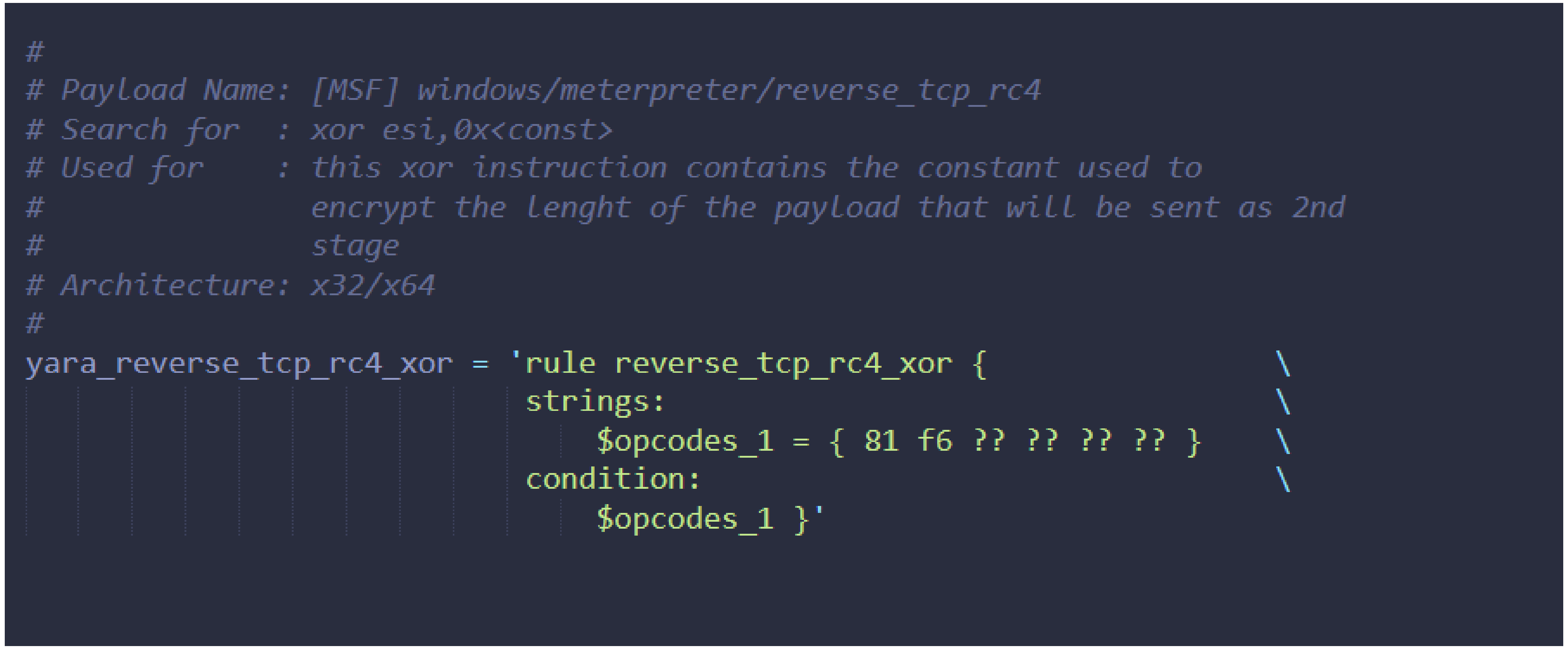
```
#
# Fixup #1
# Unicorn issue #1092 (XOR instruction executed twice)
# https://github.com/unicorn-engine/unicorn/issues/1092
#               #820 (Incorrect memory view after running self-modifying code)
# https://github.com/unicorn-engine/unicorn/issues/820
# Issue: self modfying code in the same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

Fixups examples



```
#
# 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
#
```

Implementing new rules (YARA)



```
#
# 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 }'
```

And what about CobaltStrike?

I implemented a minimal initial support for CS beacons, with the emulation and the configuration dump done if proper external tools are installed.

https://github.com/Sentinel-One/CobaltStrikeParser

Demo!

The next steps...

The tool is in beta, it works, but it needs to be extended and completed.

Contributions are not only welcome, but required, so please open Issues on the Github Repo to improve it and make it better and better. Thanks!

https://github.com/REW-sploit

Thank You!