WebAssembly
A New World Of Native Exploits On The Web
Agenda

• Introduction
• The WebAssembly Platform
• Emscripten
• Possible Exploit Scenarios
• Conclusion
Wasm: What is it good for?

- Archive.org web emulators
- Image/processing
- Video Games
- 3D Modeling
- Cryptography Libraries
- Desktop Application Ports
Wasm: Crazy Incoming

- Browsix, jslinux
- Runtime.js (Node), Nebulet
- Cervus
- eWASM
Java Applet Joke Slide

• Sandboxed
• Virtual Machine, runs its own instruction set
• Runs in your browser
• Write once, run anywhere
• In the future, will be embedded in other targets
What Is WebAssembly?

● A relatively small set of low-level instructions
  ○ Instructions are executed by browsers

● Native code can be compiled into WebAssembly
  ○ Allows web developers to take their native C/C++ code to the browser
    ■ Or Rust, or Go, or anything else that can compile to Wasm
  ○ Improved Performance Over JavaScript

● Already widely supported in the latest versions of all major browsers
  ○ Not limited to running in browsers, Wasm could be anywhere
Wasm: A Stack Machine

WebAssembly Instructions

Function Table
0x91AA7F12

Instruction Addresses
0x8F001A9B

Linear Memory

Push/Pop

The Stack
(module
  (func $add (param $x i32) (param $y i32) (result i32)
    get_local $x
    get_local $y
    i32.add)
  (export "add" (func $add)))
Linear Memory Model

Subtitle

WebAssembly Instructions

Function Table

Instruction Addresses

0x91AA7F12

Linear Memory

0x8F001A9B

Push/Pop

The Stack
(module
  (type $return_i32 (func (result i32)))
  (table 2 anyfunc) ;; creating a table with 2 places, each place takes a function of any type
  (elem (i32.const 0) $f1 $f2) ;; starting at element 0 of the table, we are adding f1 and f2
  (func $f1 (result i32)
    i32.const 42
  )
  (func $f2 (result i32)
    i32.const 13
  )

  (func $callByIdx (param $i i32) (result i32)) ;; this is the function that calls a function within the table
    (get-local $i)
    (call_indirect (type $return_i32)) ;; takes a type as an argument and pops the top value off the stack
  )
  (export "callByIdx" (func $callByIdx))
)
Wasm in the Browser

- Wasm doesn’t have access to memory, DOM, etc.
- Wasm functions can be exported to be callable from JS
- JS functions can be imported into Wasm
- Wasm’s linear memory is a JS resizable ArrayBuffer
- Memory can be shared across instances of Wasm
- Tables are accessible via JS, or can be shared to other instances of Wasm
Demo: Wasm in a nutshell
Emscripten

- Emscripten is an SDK that compiles C/C++ into .wasm binaries
- LLVM/Clang derivative
- Includes built-in C libraries, etc.
- Also produces JS and HTML code to allow easy integration into a site.
Old Exploits
Old Exploits: Integer Overflow

- Int overflows within the C code work as normal
  - Can be a gateway to other exploits or just a simple sign flip
- More interesting: JS numbers and C types and Wasm
  - Wasm: int32, int64, float32, float64
  - JS: $2^{53}-1$ (or sometimes $2^{32}-1$)
  - C: more than I can fit on this slide
Old Exploits

Integer Overflow

Demo
Old Exploits: Format String

- Right way: `printf("\%s", userstring)`
- Wrong way: `printf(userstring)`
- Extra format specifiers appear to be pulling values from linear memory
- `%n` works fine, so we can write too!
- TODO
Old Exploits

Format String

Demo
Old Exploits: Buffer Overflows

● Good
  ○ C doesn’t do bounds checking, so neither does Wasm
  ○ Overflows can overwrite interesting values
    ■ Change a privilege level, account balance, etc.

● Bad
  ○ If you overflow past your linear memory, you get a JS error
  ○ Function structure of Wasm means no call stack as we know it, no return pointers to overwrite, etc.

● Ugly
Old Exploits

BOF
Demo
Old Exploits: Et Cetera

- Probably working vulns/exploits/techniques:
  - TOC/TOU
  - Timing/side channels
  - Race conditions
  - Heap-based arbitrary writes
- Probably doesn’t work
  - UAF, null dereferencing, etc.
  - Classic buffer overflows, ROP
  - Information Leaks
New Exploits
New Exploit: BOF -> XSS

- If a value exposed to Wasm is later reflected back to JS, and there’s a traditional buffer overflow, we should be able to overwrite the reflected value
  - We use a user-tainted value to overwrite a “safe” value
  - DOM-based XSS
    - Depends on what types of variables and how they were declared
- Likely to not be caught by any standard XSS scanners, since they won’t see the reflected value as editable
- **BONUS:** JS has control of the Wasm memory, tables, and instructions, so XSS also gives us control of any running Wasm if needed.
Emscripten: New Exploits

Buffer Overflow -> XSS
New Exploit: FP Overflow

- Function pointers aren’t really “pointers” in the C sense
- Variables will store indexes to the function table
- Wasm code will say “grab the index from that variable, then call that function”
- We’ve already shown we can modify the values of some variables via overflows
- Can you see where this is going?
New Exploit: FP Overflow (2)

● Almost ROP?
  ○ Find functions you’d really like to call, but can’t, overflow the function pointers somewhere else to point to those functions
  ○ Bad news: Signatures much match
    ■ Silver lining: There are only 4 types in Wasm
● Look for useful functions within the context of the application
  ○ “transferMoney”, “changePW”, etc.
  ○ Or, just look for something that lets you run JS (maybe builtin!)
● Similar technique described by Jonathan Foote at Fastly (his is TC/Serialization-related)
Demo: FP Overwrite -> XSS
New Exploit: Server-side RCE

- All of the previous techniques can also be used against Node
- Remote Code Execution on the server
Demo: FP Overwrite -> RCE
Emscripten: Security Features

• Things that don’t matter:
  ○ Non-executable Memory (NX/DEP)
  ○ Stack Canaries

• Protections not present:
  ○ Address Space Layout Randomization (ASLR)
  ○ Library hardening (e.g. %n in format strings)

• Effective Mitigations:
  ○ Control Flow Integrity (CFI)
  ○ Function definitions and indexing (prevents ROP-style gadgets)
Application Developers

- Avoid emscripten_run_script and friends
- Run the optimizer
  - This removes automatically included functions that might have been useful for control flow attacks
- Use Control Flow Integrity
  - There is a performance penalty
- **Fix your c bugs!**
Attacker

- Look for emscripten_run_script and friends
- Use overflows or other write attacks to modify Wasm data
  - Possible XSS, can also modify the Wasm itself
  - Even if XSS is not possible, can still modify data or make arbitrary function calls in some cases
- Using these same tricks vs. Node -> RCE
More Information

Whitepaper: Security Chasms of WASM
- Tyler Lukasiewicz
- Brian McFadden
- Justin Engler

Justin Engler
justin.engler@nccgroup.trust
@justinengler

Tyler Lukasiewicz
Tyler.Lukasiewicz@nccgroup.trust
@_kablaa