Konrad Biegaj & Vineet Patel
CS: 568 | Fall 2020
Project Proposal

WASM: Bringing Back the Buffer Overflow

We propose to use Webassembly to detect and prevent malicious actions during browsing. Webassembly officially became a standard on December 5, 2019 by the World Wide Web Consortium (W3C). Webassembly (WSAM) allows for a seamless browser drop-in of code from a wide variety of languages. WSAM's stack machine is designed to be encoded in a size and load-time-efficient binary format.This means memory-intensive actions like a complex JavaScript function will run quicker without having to refactor code or increase system resources. This is done by WASM taking advantage of common hardware capabilities available on a wide range of platforms.

We are using the Everything Old is New Again: Binary Security of WebAssembly paper as the basis of our research. Specifically, we focus on the part that mentions, "...many classic vulnerabilities which, due to common mitigations, are no longer exploitable in native binaries, are completely exposed in WebAssembly."

The convenience of being able to plug C/C++ code directly into a browser with WASM is a strength; however, browser interactions operate within a different ecosystem than running native binaries on an offline system. This means almost any binary compiled by WASM may have vurnalbilites not seen in the original build. We plan on using emscripten and Wasabi to help analyze Wasm executions to find potential vulnerabilities in the stack.

One of the most significant attacks outlined in the paper revolve around buffer overflow attacks. Our proposed project is to develop a tool (using Wasabi) that can help detect buffer overflow vulnerabilities in WASM binaries. A rough idea would be to probe for possible user input, and then provide the program with input and see if a buffer overflow is possible. This type of tool may be of use to developers who use memory unsafe languages (or libraries made in memory unsafe languages)

As we are somewhat new to WASM, the first milestone would be taking a crash course in WASM and getting it installed on our machines (Week 1). From here we would have to pick a target site and/or application that runs on a site (Week 2). We should also try to replicate many of the vulnerabilities we are trying to prevent. We should ideally have a couple of example programs with an attack strategy for each of them. (Week 3).

Once we have a target group of apps or websites, we will investigate how buffer overflows can be leveraged. After a target group is established and results are gathered we will review what mitigation practices are possible.

Milestones:

Phase I

- Install WASM on our research machines
- Install Emscripten & Wasabi
- Install and configure emscripten/wasabi for analyzing WASM

Phase II

- Research and identify open-source libraries from https://awesomeopensource.com/projects/c
- Install and compile .wasm from open source
- Capture and analyze debug and logs

Phase III

- Review findings for potential attack-vectors
- Replicate attack findings

Phase IV

- Identify pertinent findings
- Review potential WASM vulnerabilities and updates from findings
- TBD

Phase V

- Start preliminary slides for presentation
- Review lapses or limitations of current scope
- TBD