September 10: Sandboxing and Goodle's Native Client

Scribe: Linda N. Lee

September 21, 2015

1 Background

1.1 Flavors of Sandboxing

A sandbox is a security mechanism for separating running programs, often used to separate untrusted elements from trusted elements. Sandboxing mechanisms have different trust models, with some leveraging a particular OS, specific hardware, or primitives such as capabilities. NaCl is unique because its sandboxing mechanism does not require OS checks since the binary code is processed and made safe, then run with user privileges.

1.2 Approaches to Browser Code Safety

Assume that you would like to run x86 code safely in the browser. The options:

1. Have trusted developers: All developers that code anything that runs in your browser are trusted. However, this is impractical trust model and additionally unsound because even trusted developers will make honest mistakes and cause program bugs.
2. Use an OS-based or hardware-based sandboxing: examples of such methods include Capsicum, pagetables, DACs, MACs, VMS, etc. The issue with these methods are that they are platform-specific, and requires trust of the OS, which is a huge trusted computing base, or the hardware, which is difficult to do from a browser standpoint.
3. Use software-based fault isolation (use NaCl!): given binary code, ensure that the code is safe by disallowing unsafe behaviors and constraining its access to the NaCl module. Since the code is made safe, then the code is allowed to run with user privileges. This approach does not require trust in developers, OS, or the hardware.

1.3 Safety of Instructions and Behaviors

There are instructions that are always safe, sometimes safe, and always unsafe. There are instructions which are always safe, such as arithmetic operations. Even an error in an arithmetic operation is safe provided that the result is not used in any way. Some instructions, such as a move or a jump, have the potential to be dangerous and should be masked. There are other calls which are just dangerous in nature for a binary to have access to. For instance, direct syscalls are deemed to be always dangerous, so they should be prohibited.

2 NaCl Overview

2.1 Why NaCl

The purpose of NaCl is to allow binary code to run in the browser without sacrificing the security that comes from higher level languages which provide support through their application programming environments. NaCl mitigates issues commonly associated with binary code by providing software fault isolation to address security issues and operating system portability for binary code to make this deployable. This allows for browser code to enjoy the benefits of binary code, such as performance, access to hardware features, and the ability to reuse legacy code or existing libraries. This is especially useful for computationally intensive operations, such as rendering gaming graphics or implementing intensive cryptography.

2.2 Using NaCl

(From the in-class demonstration of the hellowowrld NaCl module): to use NaCl, embed the NaCl module into your webpage via HTML. When the webpage starts, it runs the Nacl module. Because of NaCl's architecture, failing code in the module does not cause a crash in the browser (demonstrated by intentionally inserting a bug into the helloworld module and checking that the browser still functioned otherwise).

2.3 Threat Model

The threat model is an attacker-controlled binary which can attempt to do anything in the non-attacker controlled NaCl module on the non-attacker controlled OS. NaCl aims to contain the binary within the module boundaries and prevent unsafe instructions from executing, explicitly:

- NaCl Goal #1: remove and mask all disallowed instructions in the binary.
- NaCl Goal #2: prevent the binary from accessing memory or executing code outside of the module boundary.

3 NaCl's Architecture

3.1 Summary

NaCl communicates with the browser via IMC. It has an inner sandbox, outer sandbox, isolated exceptions, and service runtime facilities.

- The IMC provides a communication channel between the browser and the Nacl sandbox. NaCl modules are allowed to share memory and communicate as if they were a single process (see Figure 1).
- The inner sandbox (see Figure 2) enforces rules on the untrusted code and uses a static analyzer to confirm those rules. It enforces data integrity, reliable disassembly, no unsafe instructions, and control flow integrity (CFI). A data sandbox is created by combining CISC fault isolation and 80386 segmented memory. Reliable disassembly is ensured by making

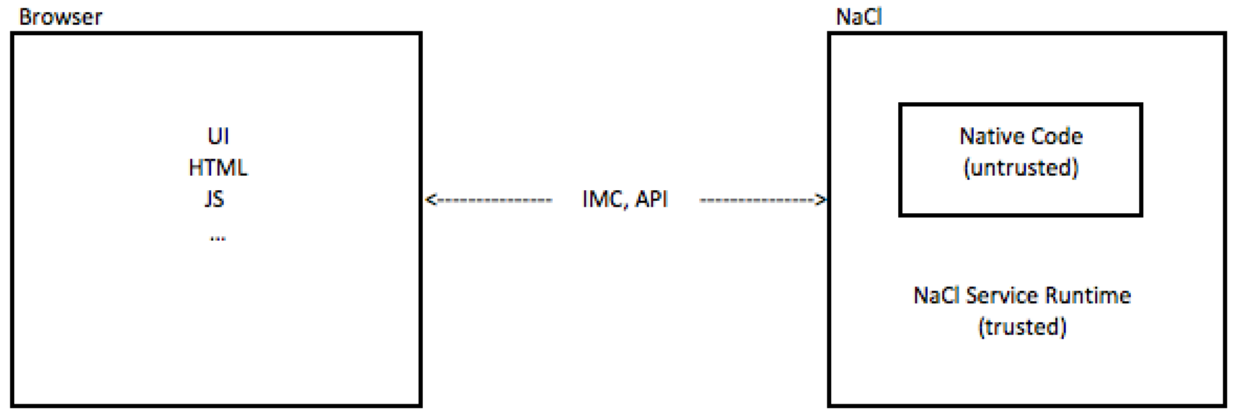

Figure 1: The NaCl architecture.

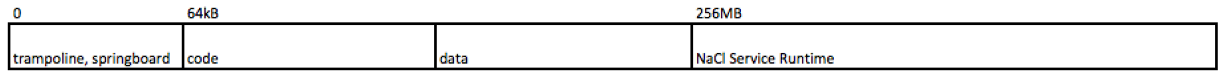

Figure 2: Sandbox Memory Layout.

binaries non-writable and valid instructions made reachable from the base address. Unsafe instructions are blacklisted. CFI is maintained by implementing its own nacljmp instruction and ensures that any address reachable from the set of instructions reachable from the start address is also in the set.

- The outer sandbox mediates system calls, double-sandboxing the binaries and providing additional security via defense in depth.
- Each NaCl module runs in its own OS process to isolate exceptions from different binaries in order to try and prevent exploits that involve exceptions.
- The service runtime helps prevent trampoline attacks by blocking system-call trampolines with a halt (hlt) instruction at the beginning of the trampoline call. Per-thread trusted stack lives outside the untrusted address space to stop threading attacks from reaching it. It prevents attacks from abusing it ability to jump outside of the binary workspace by requiring alignment to the beginning of the instruction, and therefore, the half instruction, when this is called from within the untrusted binary workspace.

4 Class Discussion

4.1 Designing for NaCl's Security Goals

In class, we discussed how NaCl could achieve its goals of providing software fault isolation. Each member of the audience provided a suggestion on how to address a goal, and the reasons for the necessity of each measure was discussed.

Recall that:

- NaCl Goal #1: remove and mask all disallowed instructions in the binary.
- NaCl Goal #2: prevent the binary from accessing memory or executing code outside of the module boundary.

Class discussion about how to achieve these goals within the NaCl architecture:

- Designing for Goal #1: inspect each instruction to check if it is a dangerous call and take appropriate actions.
- Designing for Goal #2: mark code pages and not writable, mark all data pages as not executable, don't let code jump outside the boundary, access data only within the boundary, jumps should only occur to the start of an instruction.

4.2 NaCl's Sufficient Safety Rules

The following are NaCl's rules which are considered sufficient for the binary to be safely run. Note that this is a sufficient set, and not necessarily the minimum set of rules. Some rules are independent of the NaCl implementation (e.g.: rule 1), while others are are dependent on the NaCl implementation (e.g.: rule 2).

1. Executable code is not writeable.
2. Binary statically linked at 0, code starts at 64k.
3. All computed jumps use pseudojump.
4. The binary is padded to a page boundary with at least 1 halt instruction.
5. No instruction can span the 32-byte boundary
6. All jump targets are reachable by fall-through disassembly from the start.
7. Disallow access to data outside the data segment and code outside of the code segment.

5 More on NaCl

Want to know even more? Assuming you have already read the paper, check out:

- How NaCl was received: `https://en.wikipedia.org/wiki/Google_Native_Client#Reception`
- Google's NaCl Documentation: `https://developer.chrome.com/native-client`
- Google's Technical Overview: `https://developer.chrome.com/native-client/overview`
- Implementing NaCl: `http://www.chromium.org/nativeclient`