ASLR derandomization and JIT exploits

February 10, 2014

1 Summary

In this class we discussed various mechanisms to defeat ASLR. The main point of the discussion cenetered on the observation that if you didn't use enough entropy, randomization doesn't help you much. Specifically, we talked about the ASLR implementation on Linux (PaX). Because the implementation only randomizes the beginnings of segments, you can easily pick out gadgets inside the codebase and only have to guess 2^{16} times, which ends up being easy enough to do.

The other topic of discussion focused on predicting the output of JITs. JITs end up being emitting rather boring, systematic code, that can be predicted quite easily. Our discussion focused on how this paper worked, and whether it was really relevant anyway, given that if you could exploit a browser, it already has a lot of binary code in it anyway.

2 ASLR comments

- Change the program so that the attacker doesn't know the gadgets
-
  - Mike Franz' compiler does this to compile programs with different libc versions
  - Andrew points out that with a data only attack, you can disclose a large part of the executable code and do gadget discovery online
- Adding hardware:
  - Luis pointed out aligning code on basic block boundaires and then making basic blocks fixed lengths and aligned works
  - Trade off:
    - Too small basic blocks, gives attacker more gadgets
    - Too large lots of wasted noops and possibly inefficient
    - Causes awkward changes in control flow which potentially hurts perf
  - If you want to change hardware why not just implement CFI in hardware instead
- Pointer encyption
  - Programmer wants to exploit a program by going to a ROP gadget
  - Implement using xor or something
  - But if you're using pointer encryption then the programmer doesn't know how to do that.

  - The compiled code needs to know where the key is to do things like function pointers
  - Pointer encryption: seems hard to add to legacy code,
    * Problems still exist with a TPM
    * Can recompile with same compiler, but might not have codebase

- Can you target ROP at the crash safe architecture?
- Using side channels to recover pointers
  - Andrew points out that you can use side channels (properties of the heap allocator) to learn values of pointers and learn where
  - People did this for JavaScript
- How much code do you need to get a turing compelte set of ROP gadgets?
  - Libc is turing complete, so it's fine, can do ROP attack on top of hello world
  - Optimized code has fewer gadgets, because there's a smaller amount of code
  - If you only had read and printf, then maybe you wouldn't have enough gadgets to do something evil.
  - Most interesting programs use "enough" of libc that there are probably interesting gadgets
  - Andrew points out that you can compartmentalize a program into separate modules and when you switch to a new module, you pull out the other code so that you can minimize the amount of code that can be used for ROP at any time.

3 Comments on JIT randomization

3.1 Criticism

- Why bother with discussing this technique anyway. If the attacker can exploit your browser, they can just run random code inside chrome.
- This definitely makes it easier, if you do things like ASLR within the chrome codebase, but know where the binary code is going to be.

3.2 Insights

- Machine generated fresh key for encrypting every literal, makes it so that attacker cannot guess any single literal's key and exploit it for ROP.
- This strategy maye help defend against something where you don't really know about the rest of the codebase because some extra stuff has been hardened in a finer way (e.g., using more refined CFI).
- Your CFI may allow you to jump into the generated code and may not be applied as easily to the code that you run through the JIT, so that might be a potential way to exploit a system.