Control Flow Integrity

February 11, 2014

1 Background

Control Flow Integrity (CFI) states that program execution must follow the control flow graph (CFG) generated at compile time. Traditionally, analysis performing CFI work by adding checks at run time before every indirect control transfer (i.e. a call to a function pointer). These checks verify that function pointers either are within a valid code region, are in a whitelist of pointer addresses, or are part of a valid points-to set. Direct calls do not need to be checked because it is guaranteed that they follow the CFG, and it is assumed that the code is not writable in memory due to `W⊕X`.

2 Practical Control Flow Integrity & Randomization for Binary Executables

2.1 Summary

This paper presents an alternative approach to CFI. Instead of just checking tags, the authors create a springboard that checks for integrity. Their implementation works by first decompiling binaries. They claim this is useful because it does not rely on source code or debugging information. Their binary rewriter redirects all indirect calls through their springboard, which checks whether the target location is in a white list. The springboard verifies these locations by using byte alignments and masks, which is faster than traditional CFI since tag checks are more expensive than alignment checks. They assume that all calls to `system`, `exec` and `WinExec` are direct.

2.2 Questions

Aseem: Why is `W⊕X` important for CFI?
Mike: Attackers could potentially corrupt the tags in the checks.

Alex: Why can't you overwrite the jump to springboard checks?
Mike: The paper assumes that `W⊕X` is used so these tag checks are in code and presumably can't be overwritten.

Xiao: Why is some of the performance overhead negative?
Mike: Not all operations take the same amount of time. The binary rewriting could end up improving performance. Random rewriting of binaries can influence performance by up to 8%.

Alex: Could you combine this with taint analysis?
Mike: Yes, but runtime analysis is slow.

Xiao: Since return addresses are such a problem, why don't they just put return addresses in some other stack in memory?
Aseem: There is a concept called shadow stacks that does this.
Mike: I'm not really sure how helpful it would be. What would the overhead be?

2.3 Criticisms

Andrew: You could still jump into byte aligned memory using a data only attack.

Kris: Why don't they release their code so that people can start using this in their systems?
Andrew: Their implementation is probably bad.
Mike: They might want to take money off it.

Mike: Do we need decompilers and binary rewriting?
Andrew: No, you can use a compiler backend.

They claim that ROP gadgets are no longer turing complete, however they only used Mona to find gadgets which is a weak tool.

They had to shut off JIT for IE6 and Firefox.

2.4 New Ideas

Are the tags predictable? Could they be used to create ROP gadgets?

Luis: It seems like it would be easy to implement their springboard in hardware.

3 Automated Detection of Persistent Kernel Control-Flow Attacks

3.1 Summary

This paper presents a state based control flow integrity, which approximates what CFI checks do by looking at the state of a program periodically. The

authors focused on detecting kernel root kits since they argued that root kits aim to be persistent, which CFI should be able to detect. In practice, 24 out of 25 root kits analyzed violated CFI.

The authors implemented their state based CFI as an isolated monitor in a separate virtual machine. Periodically, the monitor runs in parallel to detect persistent attacks. The period between checks is tunable so users can balance performance verse precision. The checks work by first validating the kernel text and its modules by verifying their cryptographic hashes. Then it checks for indirect calls by traversing the heap. In order to perform this last step, the algorithm needs type information so they added type annotations into the kernel code.

3.2 Questions

Aseem: Can you do a similar approach for all data pointers?
Mike: How would you make this automatic? This would require domain specific knowledge.

Matt: There is one root kit that doesn't get caught so why won't everyone use that approach?
Mike: You could check for that case manually. We don't have a good approach for data integrity.

James: Would you need special cases for each polymorphic data structure in the heap traversal algorithm?
Mike: Yes. Karla's working on a generalized solution to this with C-Strider.

3.3 Criticisms

Could potentially avoid detection by transiently getting your code to rerun.

Matt: It seems undesirable that you need an extra core for the other virtual machine and take a 6% performance hit.

Andrew: Microsoft could implement this, but it seems infeasible because it would need the source code for all third party drivers.

3.4 New Ideas

Andrew: A hypervisor is a better approach than an off board card since it is possible to lie to the card.