Control-Flow Integrity for Smartphones

Lucas Davi¹, Alexandra Dmitrienko², Manuel Egele³, Thomas Fischer⁴, Thorsten Holz⁵, Ralf Hund⁶, Stefan Nürenberger⁷, Ahmad-Reza Sadeghi¹,²

¹ University of California, Santa Barbara, United States
² Fraunhofer SIT, Darmstadt, Germany
³ Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany

Control-Flow Attacks: A Major Threat to Software Applications (on Desktop PCs and Smartphones)

The Problem of Control-Flow Attacks (Runtime Attacks)

- Control-flow attacks are possible because applications still suffer from a variety of (memory-related) vulnerabilities allowing an adversary to compromise the application flow via diverse techniques, e.g., stack overflows, heap overflows, integer overflows, pointer subterfuges or format strings.
- Recently these attacks have been applied to smartphones applications as well of which hundreds and thousands are downloaded every day.
- 2010: Stealing the user's SMS database on iOS [Iozzo et al., 2010]
- 2010: Launching a remote reverse shell on Android [Keith, 2010]
- 2011: Rooting an iOS device via a PDF-based jailbreak [comex, 2010 and 2011]

Problems of the Existing Intel x86 CFI Approach

- Requires a sophisticated binary instrumentation framework (Vulcan) that is not publicly available and only supports x86 and Windows operating systems
- Moreover, the binary rewriting approach requires debugging information that are typically not included in third-party applications

Challenges for a CFI Solution on Smartphones

- ARM and Intel x86 differ substantially
- No dedicated return instruction
- The program counter is directly accessible
- Side-Effects: Control-Flow changes may involve the loading of several other registers
- ARM supports two instructions sets (32 Bit ARM and 16 Bit THUMB) and the processor can switch among these at runtime
- More problems on iPhone:
  - Applications are encrypted and signed
  - iOS is closed-source and cannot be changed

Implementation Details of Our CFI Library

- The runtime module ensures that indirect jumps and calls follow a valid path in the CFG
- We maintain shadow stacks to hold valid or past values of return addresses
- We handle exception (e.g., stack smashing) by returning to the last non-malicious code
- Shellcode and libc attacks are mounted on a call to the generic message handling function `objc_msgSend`

Basic Principle of Control-Flow Attacks

The adversary exploits a vulnerability of a benign application at runtime (step 1) and afterwards redirects the control-flow either to injected (malicious) code (step 2a) or to existing code pieces residing in shared libraries (step 2b)

Control-Flow Integrity Framework for Smartphones

We present the design and implementation of the first CFI enforcement framework for iOS. Our framework can be divided into two phases:

1. Static Analysis: We developed new tools and extended the PiOS framework [Egele et al., NDSS 2011] to recognize all branch instructions and derive the CFG of an iOS application
2. Runtime Enforcement: We developed a new CFI library that consists of a load-time module which re-writes the application on-the-fly, and a runtime module which performs the control-flow checks

Conclusion

First CFI Enforcement Framework for Smartphones

- Requires no access to source code
- Performs binary rewriting on-the-fly and is therefore compatible to application signing/encryption and memory randomization (e.g., ASLR)
- We addressed unique challenges of smartphone platforms and operating systems
- Our CFI enforcement is efficient and induces acceptable performance overhead