Good Code, Bad Computations: a Computer Security Gray Area

Return-oriented programming shows limits of software defenses

October 28, 2008

Daniel Kane

If you want to make sure your computer or server is not tricked into undertaking malicious or undesirable behavior, it's not enough to keep bad code out of the system.

Two graduate students from UC San Diego's computer science department-Erik Buchanan and Ryan Roemer-have just published work showing that the process of building bad programs from good code using "return-oriented programming" can be automated and that this vulnerability applies to RISC computer architectures and not just the x86 architecture (which includes the vast majority of personal computers).

This new automation and generalization work from graduate students and professors from UC San Diego's Jacobs School of Engineering will be presented on October 28 at ACM's Conference on Communications and Computer Security (CCS) 2008, one of the premier academic computer security conferences.

Last year, UC San Diego computer science professor Hovav Shacham formally described how return-oriented programming could be used to force computers with the x86 architecture to behave maliciously without introducing any bad code into the system. However, the attack required painstaking construction by hand and appeared to rely a unique quirk of the x86 design.

"Most computer security defenses are based on the notion that preventing the introduction of malicious code is sufficient to protect a computer. This assumption is at the core of trusted computing, anti-virus software, and various defenses like Intel and AMD's no execute protections. There is a subtle fallacy in the logic, however: simply keeping out bad code is not sufficient to keep out bad computation," said UC San Diego computer science professor Stefan Savage, an author on the CCS 2008 paper.

Return-oriented Programming

Return-oriented programming exploits start out like more familiar attacks on computers. The attacker takes advantage of a programming error in the target system to overwrite the runtime stack and divert program execution away from the path intended by the system's designers. But instead of injecting outside code-the approach used in traditional malicious exploits-return-oriented programming enables attackers to create any kind of nasty computation or program by using just the existing code.

"You can create any kind of malicious program you can imagine-Turing complete functionality," said Shacham.

For example, a user's Web browser could be subverted to record passwords typed by the user or to send spam e-mail to all address book contacts, using only the code that makes up the browser itself.

"There is value in showing just how big of a potential problem return-oriented programming may turn out to be," said computer science graduate student Erik Buchanan.

The term "return-oriented programming" describes the fact that the "good" instructions that can be strung together in order to build malicious programs need to end with a return command. The graduate students showed that the process of building these malicious programs from good code can be largely automated by grouping sets of instructions into "gadgets" and then abstracting much of the tedious work behind a programming language and compiler.

Imagine taking a 700 page book, picking and choosing words and phrases in no particular order and then assembling a 50 page story that has nothing to do with the original book. Return-oriented programming allows you to do something similar. Here the 700 page book is the code that makes up the system being attacked-for example, the standard C-language library libc-and the story is the malicious program the attacker wishes to have executed.

"We found that return-oriented programming poses a much more general vulnerability than people initially thought," said computer science graduate student Ryan Roemer. He and Buchanan chose to study return-oriented programming for a class project after they heard Shacham outline a series of open questions in a guest lecture he gave in Savage's computer security course last winter.

Living with Return-Oriented Programming

"The threat posed by return-oriented programming, across all architectures and systems, has negative implications for an entire class of security mechanisms: those that seek to prevent malicious computation by preventing the execution of malicious code," the authors write in their CCS 2008 paper.

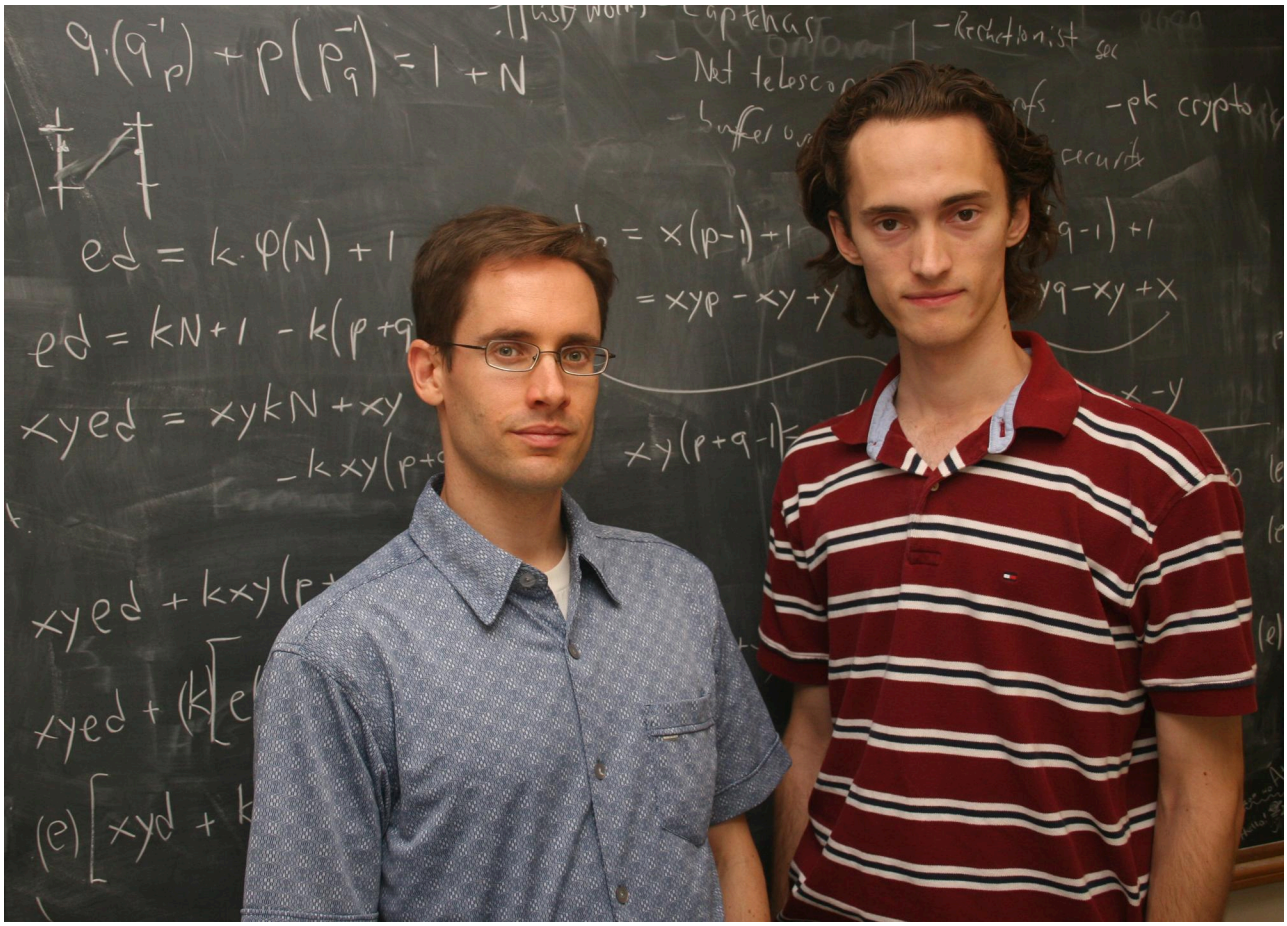
For instance, Intel and AMD have implemented security functionality into their chips (NX/XD) that prevents code from being executed from certain memory regions. Operating systems in turn use these features to prevent input data from being executed as code (e.g., Microsoft's Data Execution Prevention feature introduced in Windows XP SP2). The new research from UC San Diego, however, highlights an entire class of exploits that would not be stopped by these security measures since no malicious code is actually executed. Instead, the stack is "hijacked" and forced to run good code in bad ways.

"We have demonstrated that return-oriented exploits are practical to write, as the complexity of gadget combination is abstracted behind a programming language and compiler. Finally, we argue that this approach provides a simple bypass for the vast majority of exploitation mitigations in use today," the computer scientists write.

The authors outline a series of approaches to combat return-oriented programming. Eliminating vulnerabilities permitting control flow manipulation remains a high priority-as it has for 20 years. Other possibilities: hardware and software support for further constraining control flow and addressing the power of the return-oriented approach itself.

"Finally, if the approaches fail, we may be forced to abandon the convenient model that code is statically either good or bad, and instead focus on dynamically distinguishing whether a particular execution stream exhibits good or bad behavior," the authors write.

Media Contact: Daniel Kane, 858-534-3262


-pk crypto