CS 1699
Privacy in the Electronic Society

William Garrison
bill@cs.pitt.edu
6311 Sennott Square
http://cs.pitt.edu/~bill/1699

16: Trusting trust
Is my computer doing what I think it is?

Today we’ll discuss supply-chain attacks.

Ken Thompson’s compiler attack
• Backdoors in “trusted” entities in the system

High-profile attacks from supply chain
• Target 2013
• Home Depot 2013/2014

Best practices from SANS, CERT, NIST
Ken Thompson’s login attack

Idea: Add code into login daemon that accepts a universal password
  • e.g., Allow login if username is valid and entered password matches stored salted hash...
  • OR if password = “TheyllNeverGuessMyBackdoorPassword”

(Un?)fortunately, this hack would not last long before detection
  • How?

Instead of adding this code ourselves, let’s ask the compiler to introduce it for us
Attacking the compiler

**Insight:** A compiler can detect “login-like” code and add the backdoor automatically

- If you see a string being salted, hashed, and verified, insert a short-circuit to check for a string constant
- But how can I hide *this* backdoor?
- **Remember:** Some compilers long ago were written in assembly, but modern compilers are programmed in high-level languages
  - For C, using `lex`, `yacc`, and lots of C itself
  - Who *compiles the compilers*? Other compilers!

Let’s get meta: When compiling a compiler, insert the backdoor that automatically inserts the backdoor in “login” code

- Ship a bugged compiler binary that will compile other bugged compilers, even from clean source code
What can we do to prevent this?
Detecting a backdoor’d compiler

Idea: Compile everything from source
• How? With what compiler?

Idea: Inspect the binary compiler carefully before compiling
• With what tools? Are they trusted?
• What if Ken also inserted code in hexdump to hide the backdoors? Or in the filesystem or elsewhere in the kernel?

Idea: Download your binary compiler from someone you trust
• How? What if Ken also inserted code in the networking layer to automatically insert the backdoor when detecting a transfer of a compiler binary?
Detecting a backdoor’d compiler

Idea: Use two compilers and compare the result

• Say C and D are two binary compilers (potentially hacked, but not identically), and S is the clean source code
• C(S) and D(S) will both be valid compilers, but won’t be identical
  • Even in the absence of a backdoor, could use different optimizations, etc.
  • However, C(S) and D(S) should be functionally identical if C and D are valid compilers
• Let F=C(S) and G=D(S) be binary compilers built from S
• Since F and G are functionally identical, F(S) and G(S) should be byte-for-byte identical compilers, unless C or D was hacked
• Is this foolproof? e.g., How to compare F(S) and G(S)?
With more layers, this only gets scarier

Most CPUs today include microcode (essentially, the chip’s firmware)

• Why?
• Higher-level instructions need to be converted to actual architecture-level instructions
  • Even binaries are not the lowest level of code
• What if Ken hacked my microcode?
• e.g., Before executing sequence of instructions that look like login, insert instructions for backdoor
• How could one ever detect this?

... Should we really trust, say, Intel not to do this (or be hacked)?
Ken’s attack is a type of supply-chain attack

**General idea:** A supplier violates the trust placed in them
- In this instance, a software supplier

**Example:** Fraudulent security software
- Users trust legitimate-looking security software
- Installing actually puts them at greater risk

**Example:** Hacked security software
- This is potentially worse, as even well-informed users trust security software from vendors with strong reputation
- Consider **CCleaner** hack in September 2017
  - Software development process hacked, malware inserted before signing
Other types of supply-chain attack

Target, 2013

- Initial reconnaissance potentially from Microsoft report detailing Target’s use of their technology
- Target Supplier Portal provides specific targets
- Phishing email compromised Fazio Mechanical, a refrigeration company
- Citadel trojan installed at Fazio (some sources say weak antivirus)
- Fazio’s login details for Target’s network stolen
- Privilege escalation to gain control of Target servers (Windows)
- RAM-scraping malware installed on POS systems

Home Depot, 2013: Similar story
What other types of supply-chain attack is the average user vulnerable to?
CERT advice via SANS

Some of the mechanisms to prevent privileged insider abuse:

- Enforce separation of duties and least privilege
- Implement strict password and account management policies and practices (for all users)
  - Complete mediation
- Log, monitor, and audit employee online actions
- Use extra caution with system administrators and privileged users
  - Related to fail-safe defaults and least common mechanism

Important: Integrate suppliers with the organization’s security practices

- e.g., Two-factor for all, including contractors and suppliers
Excerpts of NIST recommendations

Within access control:
  • Access enforcement for all users
  • Information flow enforcement
  • Separation of duties
  • Least privilege

Within audit and accountability:
  • Audit review, analysis, and reporting
  • Non-repudiation
  • Monitoring for information disclosure

... and many, many more
  • Contingency plan, incident response, security training...
Conclusions

Supply-chain attacks are the result of misplaced trust
  • Supplier may be malicious or negligent

Trust has to start somewhere
  • Is my computer really doing what I think it is?
  • How many entities had a chance to backdoor, say, the average smartphone?

Security and privacy are not local problems
  • Must consider the system overall

Saltzer and Schroeder design principles keep coming up, but are not perfect

Next time: Midterm results; privacy in location sharing