Computer Security

06r. Assignment 5 review: malware

Paul Krzyzanowski

Tas: Fan Zhang, Shuo Zhang

Rutgers University

Fall 2019

Question 1 Discussion

Ken Thompson's *Reflections on Trusting Trust* paper

- Ken Thompson is the initial author of the UNIX operating system
- Currently works for Google
- This paper was presented in this 1984 Turning award speech

Basic idea:

- Modify the UNIX *login* program to add a backdoor: bypass standard password authentication and allow someone who knows about the backdoor to log in
- But people will see this backdoor code segment in the source file
  … so modify the compiler to add the code if it's compiling the login program
- But people will see that hack in the compiler
  … so modify the compiler to add this logic if it's compiling the C compiler

Question 1 Discussion

- If the C compiler detects that it is compiling the C compiler, it will add:
  - The malicious code to the C compiler to detect the compilation of the C compiler and insert this code
  - The malicious code to detect the compilation of login.c and insert bug 1
- After you compile the C compiler, you can remove the detection/bug code from the compiler source and it will be magically re-inserted each time you recompile the compiler

Anyone inspecting the source code to the compiler or login.c will not see any malicious code!

"No amount of source-level verification or scrutiny will protect you from using untrusted code."

Question 1 (a, b)

Ken Thompson's *Reflections on Trusting Trust* paper

(a) What does the match for *pattern1* test?

It matches the part of the login program that validates a user's identity.

(b) What does "bug 1" do?

Bug 1 detects a compilation of the login program and inserts code to allow bypassing a password check.

This allows the compiler to detect that it is compiling the *login* program (which is run to allow users to log into a system).

The compiler then adds code to bypass the password check.

⇒ <u>You can examine the source code to *login.c* and see nothing suspicious!</u>

Question 1 (c, d)

Ken Thompson's *Reflections on Trusting Trust* paper

(c) What is the purpose of the second pattern match in the C compiler – the one that inserts "bug 2"?

Bug 2 matches a pattern that detects the compilation of the C compiler

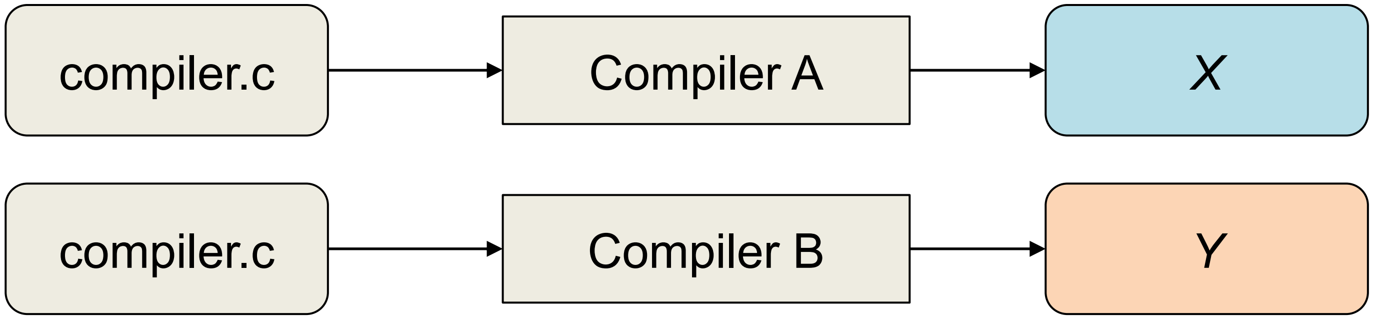
(d) What does "bug 2" accomplish?

Bug 2 inserts BOTH Trojan horses into the compiler so the source does not contain any code that matches patterns in either the C compiler or the login program.

Question 2

What does David A. Wheeler propose as a test to determine if a compiler has been made untrustworthy?

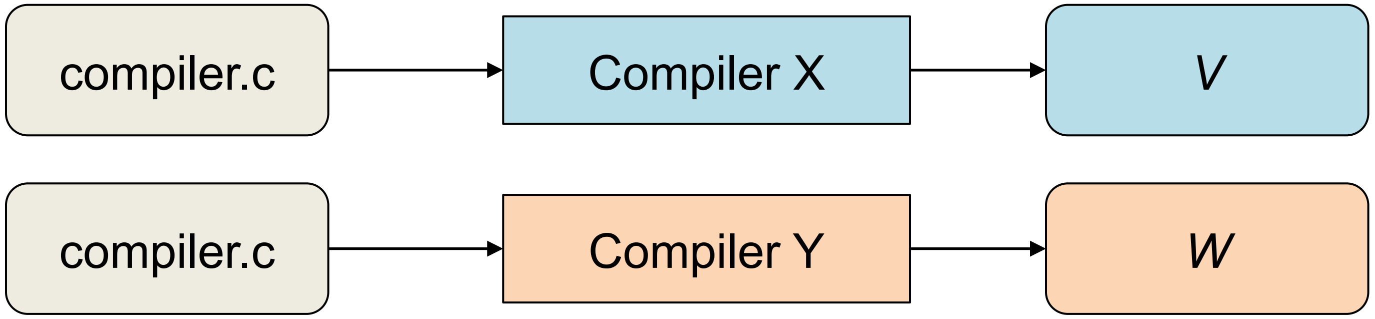
Compile the compiler source with two different compilers, producing two executable compilers, X and *Y*



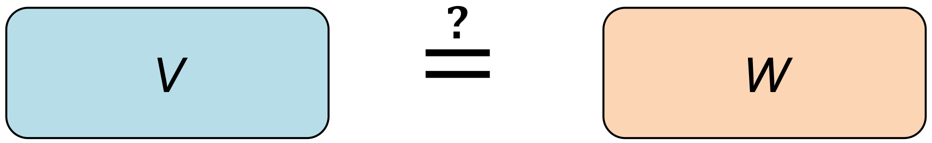
Since we're using two different compilers, *X* and *Y* will not be bit-for bit equivalent, but ***X* and *Y* will be functionally equivalent**

Question 2

- Now compile the original compiler source with these two compilers, *X* and *Y*, to produce two additional executable compilers, *V* and *W*



- Since the logic of the compiler is identical and X and Y are functionally equivalent, the resulting code produced by them, V and W, should be bit-for-bit equivalent.



- If *V* and *W* are not the same then we know that one of the compilers we used initially contains the trojan horse code and inserts a backdoor.

Question 3

What is the primary disadvantage of signature-based malware detection?

From Morton Christiansen, *Bypassing Malware Defenses*, SANS Institute Information Security Reading Room, May 7, 2010.

"The disadvantage with this approach is its ineffectiveness in detecting new variants of an old piece of malware."

Malware researchers & anti-malware companies collect examples of known malware.

There are often multiple versions of similar malware. It would be too cumbersome to download and match every version and to match the entire malware code base.

Anti-malware companies try to find code segments that are shared by multiple pieces of malware.

A "signature" is a set of bytes in the malware that we think are unique to a particular piece of malware: if we see them then we're pretty sure it's dangerous code. It's far more efficient (and safer) than having a version of every known piece of malware in your system.

The problem is that if someone creates new code, it will not match other signatures.

Question 4

What are three techniques that malware packers use?

1. **XORing the malware**
   – This is an example of a simple stream cipher. The key determines which bits get flipped.
2. **Compression of malware**
   – Compression removes information redundancy and, in the process, obscures the content.
3. **Encryption of malware**
   – Encryption changes the content (malware payload).

From the paper:

"Packers may be used to pack an executable using techniques ranging from simple XORing of the malware to compression and even encryption hereof. The malware is then unpacked during runtime."

Question 5

How does the use of packers make it more difficult to detect malware?

A packer changes the representation of the malware code. This usually results in anti-virus software being unable to detect it.

1. ***XORing the malware***

   The key determines which bits get flipped. Unless you decode the content, you cannot search for a signature.

2. ***Compression of malware***

   You cannot search for a specific bit pattern unless you decompress the data … or have signatures for the output of any of several dozen common compression algorithms.

3. ***Encryption of malware***

   You cannot scan for a signature unless you know the key and can decrypt it. The key is typically somewhere in the unpacking code (or may be downloaded from the network).

The best you can do is detect the presence of an unpacker

Question 6

What technique does the author discuss as a possible mechanism for malware to communicate with a server if it has no direct access to the Internet?

Use a DNS query (to the local DNS resolver) that will contact the adversary's DNS server and return encoded commands instead of legitimate addresses.

The malware can do a domain name lookup (e.g., look up **`secret.pk.org`**).

The DNS server within the organization will make a series of requests over the Internet, ultimately contacting the name server in charge of **`pk.org`**, which is run by the adversary

The adversary can return any encoded data – it doesn't have to be a valid IP address.

The malware can query any domain – it doesn't have to be a valid system (e.g., **`here_is_my_response.pk.org`**).

The end