Programming Languages and Trust

Veit Heller

March 21, 2019

Internet Security Meetup Berlin

Trusting Trust

In 1984, Ken Thompson was rightfully awarded the Turing Award.

Trusting Trust

He wrote a three-page paper on a scary idea: malicious compilers.

The idea

Have you heard about Quines? They are self-replicating programs.

The idea

Have you heard about bootstrapping compilers? They are compilers that can compile themselves.

The idea

What if we compile a “bugged” version of our compiler and ship it?

The idea

"The actual bug I planted in the compiler would match code in the UNIX "login" command. The replacement code would miscompile the login command so that it would accept either the intended encrypted password or a particular known password. Thus if this code were installed in binary and the binary were used to compile the login command, I could log into that system as any user."

— Ken Thompson, Reflections on Trusting Trust, page 3

The idea

"Such blatant code would not go undetected for long. Even the most casual perusal of the source of the C compiler would raise suspicions."

— Ken Thompson, Reflections on Trusting Trust, page 3

The idea

"This [second approach] simply adds a second Trojan horse to the one that already exists. The second pattern is aimed at the C compiler. [...] First we compile the modified source with the normal C compiler to produce a bugged binary. We install this binary as the official C. We can now remove the bugs from the source of the compiler and the new binary will reinsert the bugs whenever it is compiled. Of course, the login command will remain bugged with no trace in source anywhere."

— Ken Thompson, Reflections on Trusting Trust, page 3

Discussion

- It has historically mostly been regarded as a scary practical joke by compiler engineers.
- As compilers get more complex (and more modular), a malicious “optimization” pass can ever more easily be inserted into the compiler.
- If someone pointed at an obscure piece of assembly and told you that “this simple optimization buys us a 10% speed gain on auto-vectorized code in ARM”, would you just believe them?

Caveat

Nota Bene: There is a workaround described by David A. Wheeler, but it assumes that there is no non-determinism in a compiler, but there often is (see, for instance, GCC's `-fguess-branch-probability`).

$\Rightarrow$ Your best bet is probably reproducible builds.

Enter Michael Arntzenius

Aside: if you want to play with this, Michael Arntzenius has a great project on Github with an exceptional README that guides you through the process of compiling a malicious compiler called rotten.

In 2015, a lot of malware appeared on the Chinese app market.

XCodeGhost

- Downloading XCode from China is slow. There was a mirror in China that was faster.
- It was owned by malicious actors.

XCodeGhost

They inserted malicious code in the core frameworks compiled into every app built using XCode (most notably device and display classes).

Conclusio

- Maybe this attack is not part of your threat model, and that's alright.
- What comes back to bite you is your "unknown unknowns"—things you don't know that you don't know.
- Informed decisions are probably the most important puzzle piece in creating a more secure world.

References

- These slides: `https://github.com/hellerve/talks`
- This talk, but longer, and at Datengarten: `https://api.media.ccc.de/v/dg-96`
- Ken Thompson, Reflections on Trusting Trust: `https://www.win.tue.nl/ aeb/linux/hh/thompson/trust.html`
- Michael Arntzenius' reflections on Trusting Trust: `https://github.com/rntz/rotten`
- David A. Wheeler, Fully Countering Trusting Trust through Diverse Double-Compiling (DDC)—Countering Trojan Horse attacks on Compilers: `https://dwheeler.com/trusting-trust/`
- FireEye on XCodeGhost: `https://www.fireeye.com/blog/threat-research/2015/11/xcodeghost_s_a_new`

Thank you!

Questions?

Slides at `https://github.com/hellerve/talks`