RSA Conference2015

San Francisco | April 20-24 | Moscone Center

CHANGE

Challenge today's security thinking

SESSION ID: ASD-F02

Secure Application Development with Go

Nick Galbreath

CTO / Founder
Signal Sciences
@ngalbreath

SIGNAL SCIENCES

#RSAC

Today!

- How I discovered go (or who am I and why I am here)
- What is go?
- Tooling around go (yeah it's important for security)
- Dependency management
- Memory and Concurrency
- Web Applications
- Cryptography
- Wrapping up and next steps

Who am I?

- CTO and Founder of Signal Sciences.
- Author of popular SQLi detection library (libinjection - in C)
- Author of base64 code that is used in Chrome browser (in C)
- Author of "Cryptography for Internet and Database Applications" (Java-based)
- Author of social networking patents (now owned by Facebook)
- Previous professional programming in C, C++, Java, Python, PHP, typically in high-scale, massive deployments.

RSAConference2015

San Francisco | April 20-24 | Moscone Center

How did I find Go?

#RSAC

LASCON 2013

- In the LASCON 2013 Keynote, I spoke on how C (in one shape or another) is responsible for the vast majority of server CVEs.
- Still haven't done the math on this, but just a quick glance validates this.
- Can we shift C security programing problems into an optimization problem? i.e. can we get the benefits of C without the security problems?

"Any suitably creative backdoor is indistinguishable from a C programming bug". -Me (Nicholas Weaver)

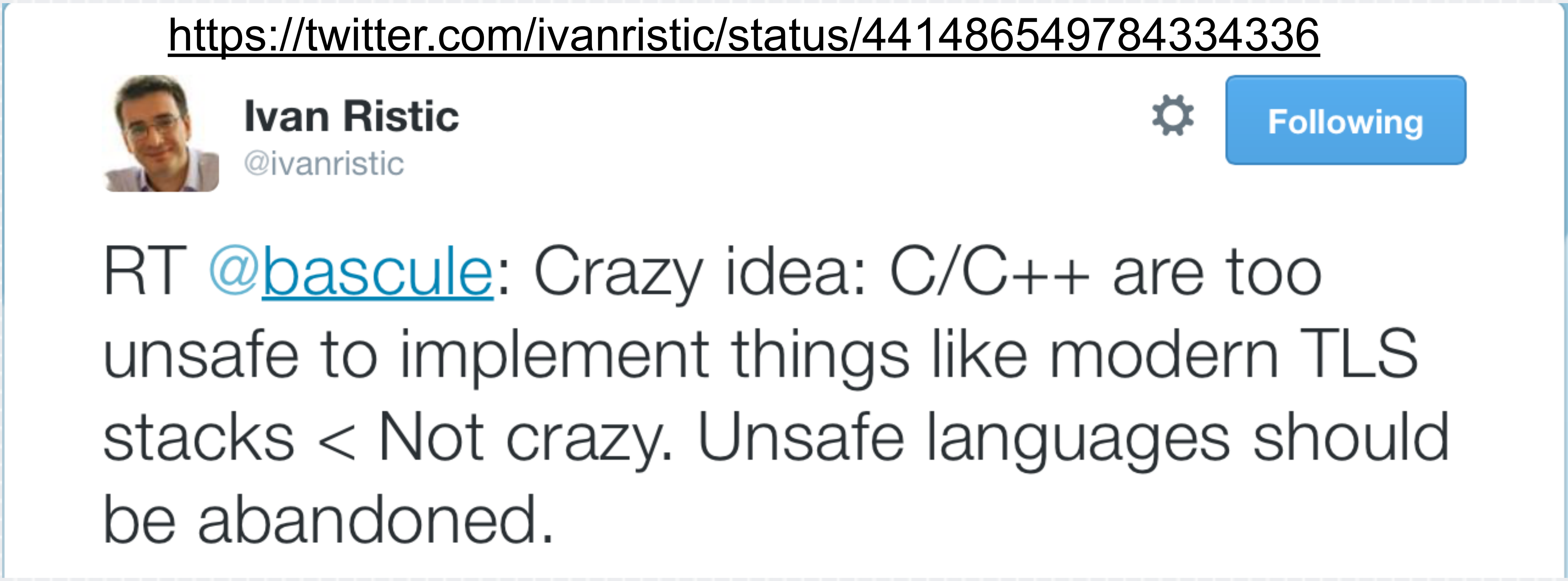
New Project… in

- But not long after that, I started a new *product*, to be installed on customers infrastructure.
- Picked C, because I could not find an alternative. It's small, fast, and easy to install the end result (single binary).
- Alternatives:
  - Not scripting language: too slow, too hard to bundle correctly.
  - Not Java: requires Java which may or may not be installed
  - Not C++: worried about complexity, and core libraries needed were in C.
  - Newer languages seem immature

But nervous about this.

- Memory errors are deadly
  (if not for security, then definitely for stability)
- Hard to find good talent
- A real barrier to rest of the team
  (will I be the only one to make changes?)
- Requires a lot of magic and tooling to really get right.
- How can a customer audit our application?

Not Crazy


https://twitter.com/ivanristic/status/441486549784334336

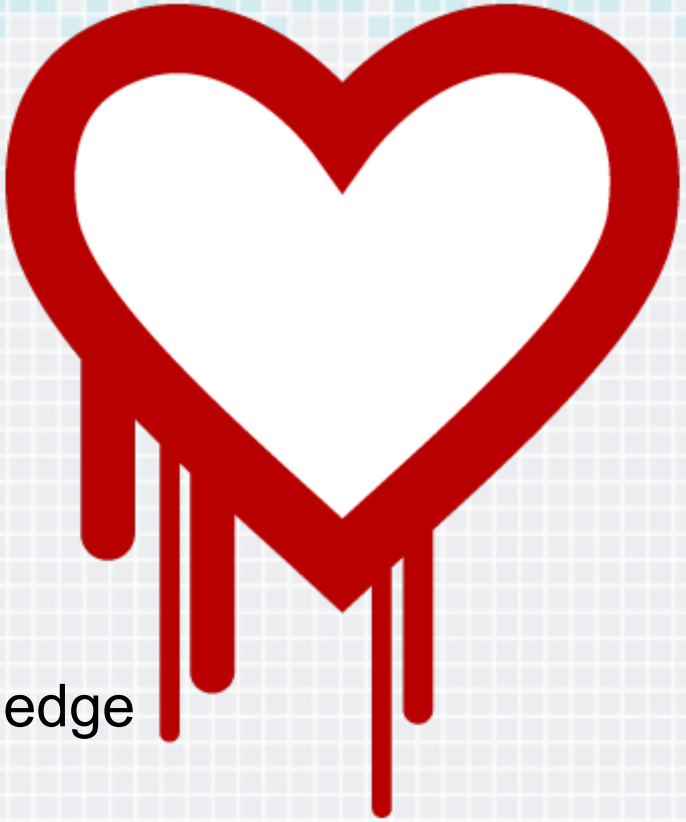
Ivan Ristic
@ivanristic

RT @bascule: Crazy idea: C/C++ are too unsafe to implement things like modern TLS stacks < Not crazy. Unsafe languages should be abandoned.

5:12 PM - 6 Mar 2014

Last Straw - April 2014

- OpenSSL Heartbleed
- We need to get off C now.
- But… in what?
  - Been hearing rumblings of Go from everything to servers to operations code
  - Looked into it more and found out many initial reactions where based on incorrect knowledge



But can I rewrite it?

- Finished about half of the online tutorial.
- Rewrote prototype of app in "long weekend"
  - Wrote "C" in golang (not very go-like but worked)
- Replaced 7 C libraries with built-in standard library
- Performance was competitive with C
- Drastically less code
- Core dump free
  - AND HERE WE ARE.

RSA Conference2015

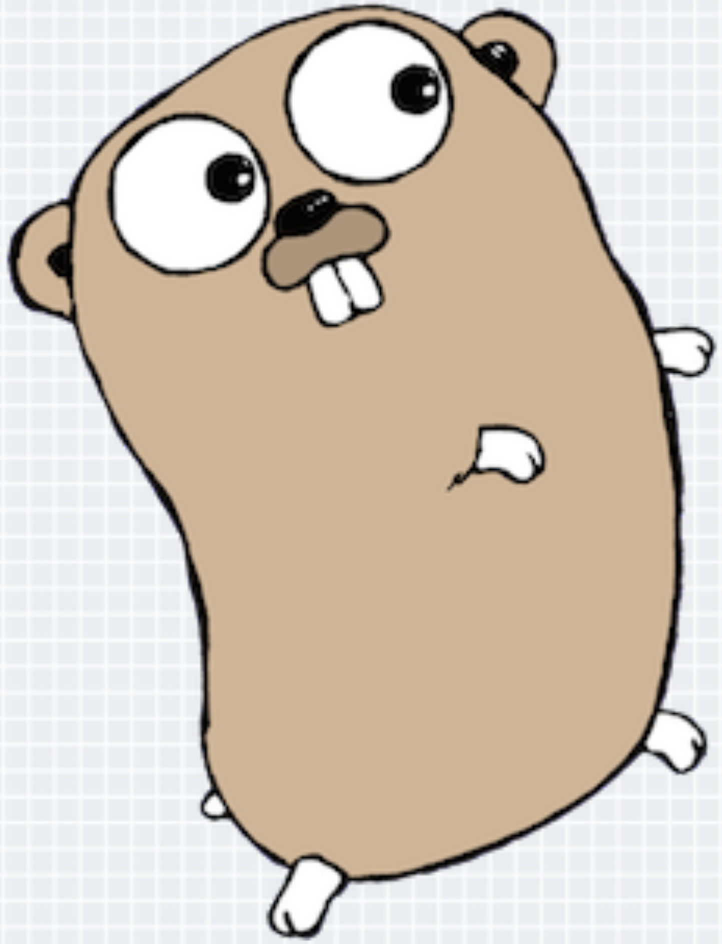
San Francisco | April 20-24 | Moscone Center

What is Golang?

#RSAC

TLDR GO

- Compiled
- Statically Typed
- Memory Managed
- Natively Executed
- Concurrent Execution



Simple?

- It's a simple language, like C. It kinda *looks like* C/C++/Java
- Perhaps too simple?
  - No Generics
  - No Operator Overloading
  - No OO-style inheritance (although it can be somewhat simulated)

"Go is not meant to innovate programming theory. It's meant to innovate programming practice." – Samuel Tesla

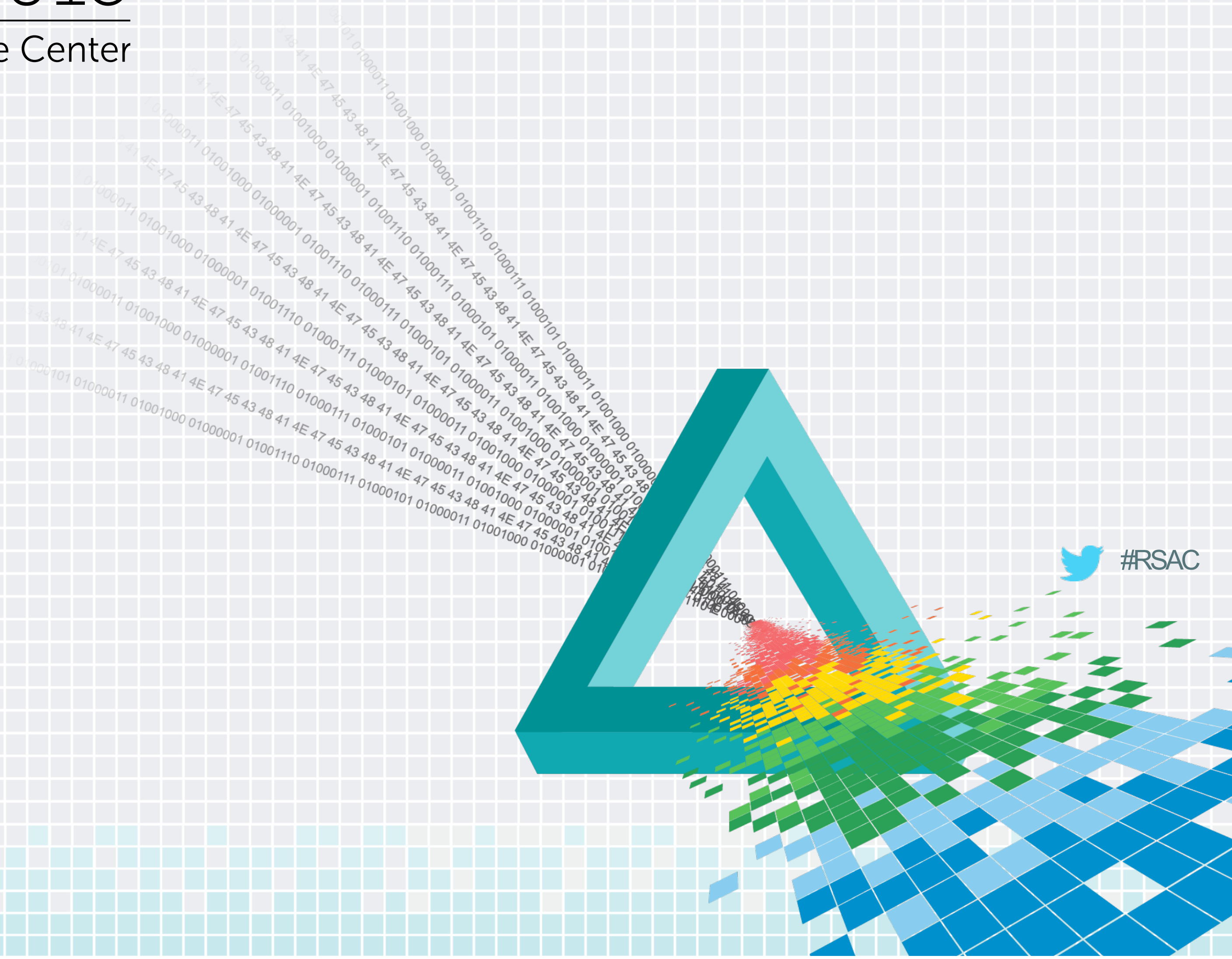
With Benefits

- Low "Dark Corners"
- Explicit is favored of implicit. This *may* mean more typing sometimes.
- While this can be maddening to *individual programmers* it makes code easier *for a team* to read.
- **This makes it easy to security audit as well.**

RSA®Conference2015

San Francisco | April 20-24 | Moscone Center

Tooling



#RSAC

What does tooling have to do with Security?

- Betters tools ⇒ better code, happier developers
- Better code ⇒ better quality
- Better quality ⇒ better security *and/or* easier to audit to make secure.

Code Formating

- Standard code formatting (and tools to reformat) are standard and out of the box.
- The result is code by different authors *looks the same.*
- Compare to say, C++ or perl, where code can look completely different between packages (not just style, but structure and format).

Code Linting and Static Analysis

- Again, standardizing how code looks make code reviews and audits focus on *real issues*.
- Go comes with a numerous static analysis tools as well. While the severity of the issues found is minor compared to say… static analysis for C, it helps prevent problems before they hit production.

Code Coverage, Allocations, Performance

- Standard unit test framework (not like xUnit!)
- Code Coverage
- Memory allocations
- Performance
- Race condition analyzer

All Written in Go

- All of these tools are written in go.
- Easy to write your own rule or checker for unsafe constructs
- Or to do code rewriting and refactoring.

RSA®Conference2015

San Francisco | April 20-24 | Moscone Center

#RSAC

Dependency Management

What does dependency management have to with security?

- "you are only as secure as your weakness dependency"
- as we have learned the hard way (i.e. heartbleed), your code may be secure, but your linked libraries may not be.

The End is a Single Binary

- A single binary is the result of a successful go build process
- No dependencies on the host (almost.. perhaps a few for virtual memory or whatnot, but not libc)
- Go is not required to run
- Completely stand-alone.
- All run-time dependencies are done at *compile time*.

Benefits

- This eliminates an enormous amount of "dev ops" work in keeping the production system stable and correct to what the developer needs to use.
- Compare to say, Chef Client which has to bundle an entire copy of Ruby since dependency management on the host is a mess.
- Deployment is simplified.. Its just one binary and don't need to upgrade OS to deploy application.

Negatives

- If some library or the golang engine has a security problem, you cannot update some shared library and magically all applications will be secured.
- You must recompile and redeploy
- For enterprise development this isn't so bad, but its not so good for OS-packaging managers (e.g. Debian, RedHat, etc).
- Makes inventory of components ("what is running now") harder

Script Language Replacement

- The lack of run-time dependencies makes golang ideal for "scripting".
- Compile time is fast (similar to Java if not better)
- Start up time is instantaneous (unlike Java).
- In spite of being compiled, and statically typed, the simplicity makes it easy to write small "scripts" you'd normally do in python/ruby/etc,
- And since its compiled and typed, entire categories of errors *vanish.*
- *(you can run go code directly by "go run file.go" as well)*

Compile-Time Dependencies

- You specify dependencies using "import" statements
- You use a fully qualified path to a source repository (many RCS supported)
- import "github.com/client9/xyxyx"
- You install dependencies locally using "go get ./…" command
- This pulls the latest version
- There is no support for specific tags, revisions, or branches!

godep

- Dependency management is solved by actually copying exactly what source revision/branch/tag/release you want into *your* source tree.
- This can be done automatically using the tool *godep,* which also preserves the meta data about revisions, etc.
- This means anyone on the team can get the exact same source code, with exactly one network call.
- Building code does not depend on an external resource being available.

Pushes responsibility onto development

- Development needs to monitor their dependencies for security announcements and update the embedded packages.
- Sounds hard, but in practice, not complicated and not hard (especially since many will update packages regularly anyways).
- May require some additional tooling to make sure dependencies are up to date.

Creates a nice split between OS Security and Application Security

- Ops can update OS packages, knowing there will be no conflict with development.
- Development is able to use what they want, without causing OS changes.

- Interesting in that….

Interesting similarity to Docker

- A docker container at the end of run is a standalone "container" with no dependency on the host.
- All run-time dependencies are done at compile time.
- Provides the same Dev/Ops split as the Go
  - Golang dependency management is for the application code
  - Docker dependency management is for the application environment (which might be near nothing with a go binary)

RSAConference2015

San Francisco | April 20-24 | Moscone Center

Memory

#RSAC

Memory Safety

- Golang is a memory-safe language, similar to say java.
- Garbage collection frees unused memory
- All array references are range checked, and if out of range, a panic occurs (kinda like a core dump on the existing thread.. sorta).
  - If you aren't careful a DoS attack could exist due to panics, but not a security problem *per se.*

Control of Memory

- Unlike Java
  - you control of size and order of structures.
  - you control if allocation is on the stack (cheapo) or on the heap (requires garbage collection)
- You control pass-by-value (a copy) and pass-by-reference (a pointer to an existing object)

Pointers, without Pointer Math

Dave Cheney @davecheney Jul 11
From @francesc's Go for Javaistas talk

No pointer arithmetic, no pointers to unsafe memory.

```
a := "hello"
p := &a
p += 4  // no, you can't
```

https://twitter.com/davecheney/status/487506186737319936

Unsafe if required

- One can integrate with C-libraries, C-types, java, and SWIG by using the "unsafe" pointer type and / or "cgo"
  - http://golang.org/pkg/unsafe/
  - http://golang.org/cmd/cgo/
- Easy to audit for (and find alternatives)
- In practice I don't see this used too often. The Go community mostly has eliminated the need for specialized C libraries.

Concurrency

- Golang Concurrency could be a whole other talk
- Uses *goroutines* which are "light weight user threads" or something similar to "coroutines" and *channels* which perhaps best described as in-memory queues or similar to Unix pipes.
- In general *much* simpler than POSIX threads
- But, they can cause race conditions possibly corrupting a *value* in memory (but not corrupt a pointer).
- Golang does provide a race-condition detector, and a full suite of mutex, read-write locks.
- In practice you'll find the use of mutexes to be limited.

Looking for security trouble spots in Go code

By Scott Piper 2015-04-15, http://bit.ly/1zRtsl9, concluded:

Conclusion

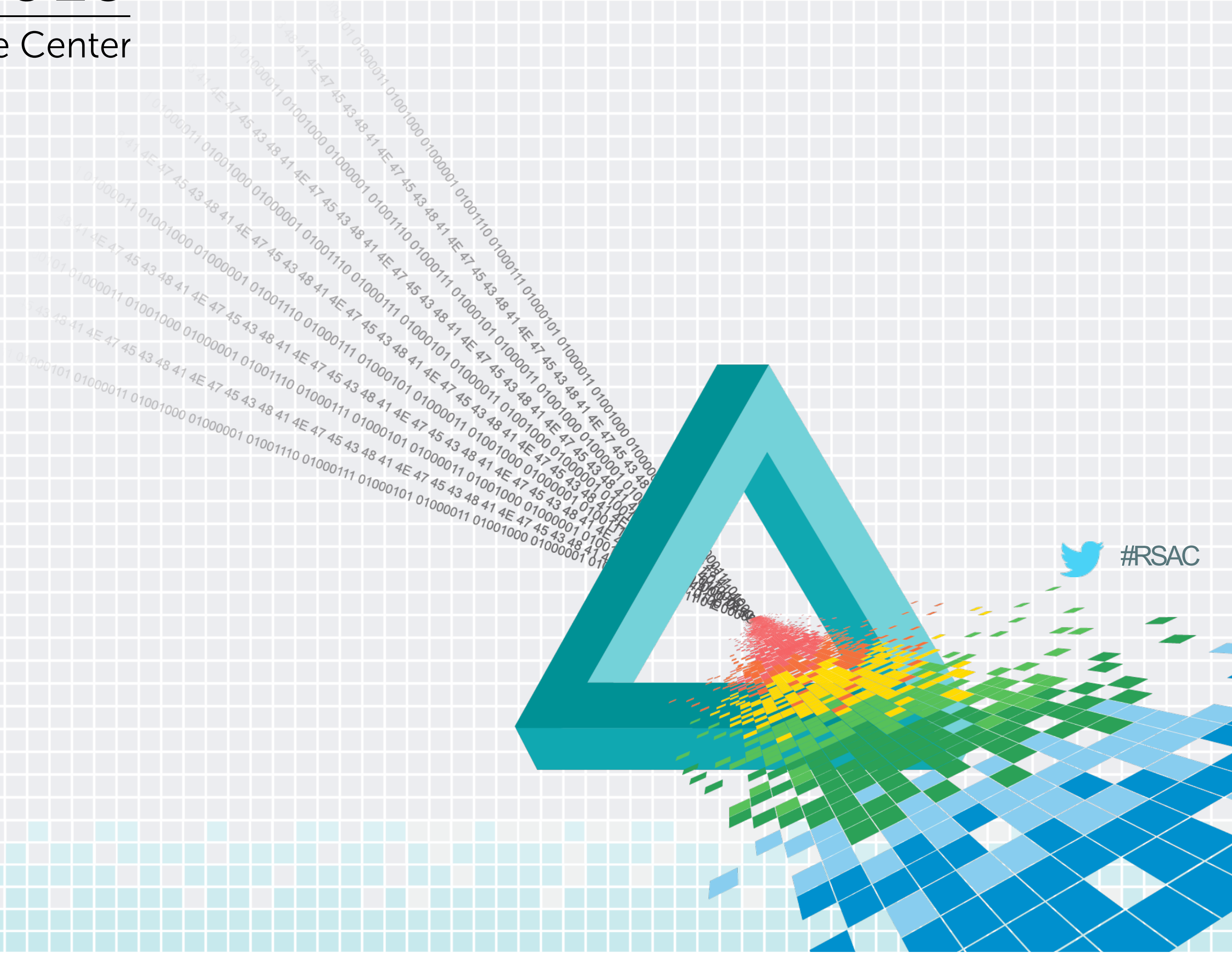
Unless you are using compiled libraries (derived from C code or other languages that aren't memory safe), you are safe from buffer overflows, use-after-free's, and other memory safety bug classes.

RSAConference2015

San Francisco | April 20-24 | Moscone Center

Security for Web Applications

#RSAC



XSS

- Golang's templating system is HTML *context-sensitive* and performs the correct escaping / encoding for the data being inserted into the document.
- This means out of the box, the templating *is secure by default.*
- One can explicitly insert HTML, and certainly some bugs might exist, but overall, *for the application developer* it's mostly invisible.
- Good summary of technology by Isaac Dawson of Veracode: http://www.veracode.com/blog/2013/12/golangs-context-aware-html-templates

One Gotcha

- Must use import **`html`**`/template`, not **`text`**`/template`.
- Can't tell which is used without inspection since both are invoked as `template.Exec(…)`
- If someone changes `html/template` to `text/template` your application runs, works the same, but is wide open to XSS.
- Learned this the hard way with the team learning Go.
- Recommend you add a unit test that renders a bogus page, and check that XSS protections are done correctly.

SQL and SQLi

- Nothing new here.
- Prepared and Parameterized queries are similar to other languages and their SQL drivers
- http://stackoverflow.com/questions/26345318/of-golang-database-sql-and-injection

As long as you're using Prepare or Query, you're safe.

```
// this is safe
db.Query("SELECT name FROM users WHERE age=?", req.FormValue("age"))
// this allows sql injection.
db.Query("SELECT name FROM users WHERE age=" + req.FormValue("age"))
```

Mongo and Mgo

- Of note is the main (third party) mongo driver.
- Extremely difficult to perform an injection as raw queries are unusual and difficult to do.

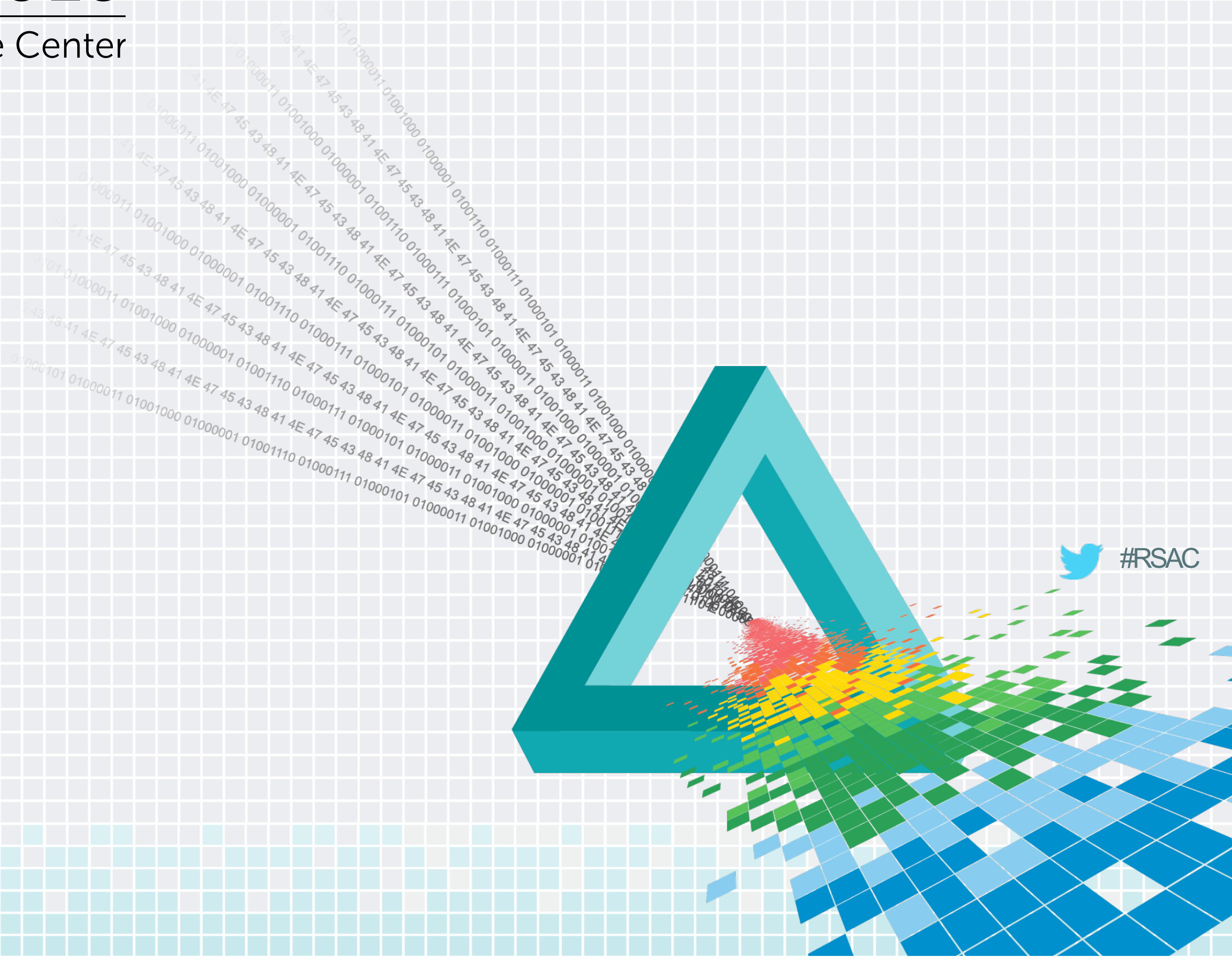
Hash DoS and Web Programming

- Built-in protections for Hash DoS
- http://www.ocert.org/advisories/ocert-2011-003.html
- Golang Maps start with an initial random seed

RSA®Conference2015

San Francisco | April 20-24 | Moscone Center

Cryptography



#RSAC

Basics

- Full built-in suite of basic cryptographic functions
- Cryptographic hashes
  - SHA1 and 2
  - HMACs
- Secret key cryptography
  - AES
  - DES
- Public Key Cryptography
  - DH
  - RSA
  - Elliptic Curve

Cryptography TLS

- Full TLS suite
- 100% go - not OpenSSL (or other C code based).
- Works. No need for additional proxy.
- Reads certs from from host OS, but can directly specify root certs.
- Various issue still being worked out… mostly dealing with "real-world" (aka broken) clients and servers in the wild (although the author has not had any issues with it).
- Not Audited! But what is?

Bonus Cryptography

- Algorithms under development or not part of standards are often included in an external package https://godoc.org/golang.org/x/crypto
- Of note particular note:
  - OpenPGP
  - SSH client and server
  - NaCL (High level crypto interface)

Random Numbers

- Unfortunately non-cryptographic "random" numbers are in the package:
  http://golang.org/pkg/***math***/rand/
- Cryptographic quality numbers are under:
  http://golang.org/pkg/***crypto***/rand/
- Fortunately, they implement different interfaces, meaning if you change "import "crypto/rand" to "import math/rand" the code will not compile.
- Unlikely to get them mixed up, but the use of same name is unfortunate IMHO

RSA Conference2015

San Francisco | April 20-24 | Moscone Center

#RSAC

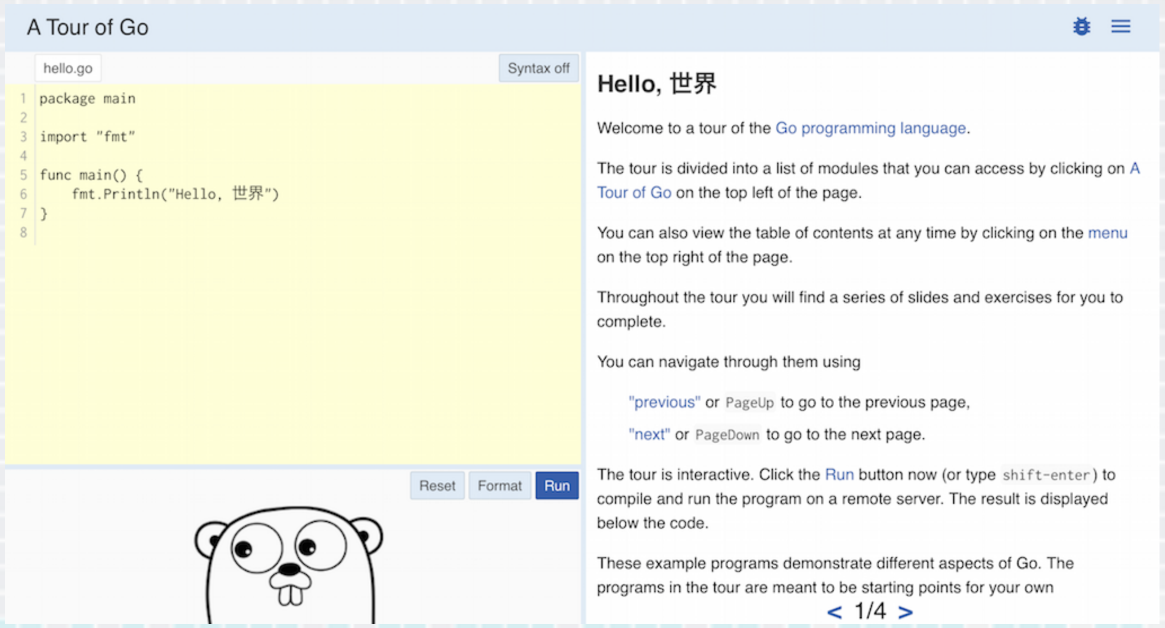
Conclusion and Next Steps

Conclusion

- If I didn't like go, I wouldn't be speaking.
- Love the performance and single binary output
- Standard library is fantastic but equally so the golang ecosystem of 3rd party libraries.
- Had people from ruby, java, and C become productive very quickly
- Everyone goes through "*why doesn't Go have* ____" or attempts to write Go in the style of their previously favorite language. It takes a while to learn the "go way" of doing this.
- Recommend!

Getting started, take the Tour

- http://tour.golang.org/ online tutorial.
- Its a very good way to learn go, and try it out, without installing anything.   It's all interactive in the browser.



```
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

Hello, 世界

Welcome to a tour of the Go programming language.

The tour is divided into a list of modules that you can access by clicking on A Tour of Go on the top left of the page.

You can also view the table of contents at any time by clicking on the menu on the top right of the page.

Throughout the tour you will find a series of slides and exercises for you to complete.

You can navigate through them using

"previous" or `PageUp` to go to the previous page,

"next" or `PageDown` to go to the next page.

The tour is interactive. Click the Run button now (or type `shift-enter`) to compile and run the program on a remote server. The result is displayed below the code.

These example programs demonstrate different aspects of Go. The programs in the tour are meant to be starting points for your own

< 1/4 >

Identify your needs

- Improve productivity?
- Replace unsafe C code?
- Have 4 different ruby versions running?
- Too many ways of installing python packages?
- Faster startup time?
- Is production deployment to complicated or too slow?

Pick a sample to convert or rewrite

- Pick the smallest test case you can
- But add in steps to do
  - go fmt
  - golint
  - go vet
  - Set up unit test and code coverage in upfront.
  - *on every commit*
- All these tools are out of box, so it easy to make a build pipeline.

Try it out, then

Go Enjoy!

SIGNAL SCIENCES

Nick Galbreath @ngalbreath nickg@signalsciences.com