The Whole World is Burning

Charles L. Yost

2017-01

Speaker Bio

Charles Yost is currently a Security Developer at Binary Defense Systems. He has worked in the IT industry for over a decade in a wide variety of roles including: Printer Technician, VoIP Systems Administrator, .Net Developer, Web Developer, and Security Developer. Throughout life his number one passion has been learning new skills. He can often be found researching a topic, attempting to keep up with the quickly evolving field of technology. Charles enjoys teaching and talking to others about technology. He is a member of NEOISF, and attends as many conferences as he can justify with his wife.

Binary Defense

Binary Defense Systems is a local Managed Security Service Provider. We provide a full range of MSSP services including: monitoring, consulting, threat intelligence, and incident response. We also are the creators of Vision, a real-time software solution which gives organizations immediate visibility into the latest attack vectors. Contact us at `https://binarydefense.com/`

Speaker Contact

Twitter: @CHARLESLYOST

GitHub & YouTube: Yoshi325

This Talk:
`https://github.com/Yoshi325/talks-bucket-of-water`

Summary

Information Security is a galvanizing term. It carries much power, and therefore, much responsibility. It can be hard to obtain buy-in for long term security needs from Management. And with all the vulerabilities cropping up every day, the pressure to “be secure” can be overwhelming.

Summary

How can a lone developer make a difference? This talk is all about the second half of its title. Common security concerns will be reviewed, then addressed with a focus on what can be done when faced with them or how to avoid them in the first place. The whole world is burning. But you can make a difference.

Here is your bucket of water.

This talk is NOT

OSWASP TOP 10

An in-depth look at all aspects of security

An introduction to secure programming

This talk IS

Make some security choices easy(er)

When there is not a clear choice,

compare and contrast the choices

Spans Sysadmin and Developer scope

Web focused at first, transitions

to broader development.

Let’s Go!

Transport

- SSL is broken
- Use TLS, 1.2 if possible

Transport Configuration

Cipherli.st Strong Ciphers Configurations

Mozilla SSL Configuration Generator

Certificates are Expensive!

- Processing

Certificates are Expensive!

- Money

Let’s Encrypt

Let’s Encrypt is a free, automated, and open Certificate Authority.

How about Free?

Any Cons?

Domain Validation

What if I need better?

Buy extended validation (EV) certificates for public web servers.

Yes, the are expensive.

Access

- Identity
- Authentication
- Authorization

Identity

Often, username.

Authentication

Validation of the right to use an identity.

Authorization

Validation of an identity's right to use a resource.

TLS Client Certificates

- Rare, but useful.
- After install, seemless authentication.

HTTP Basic Authentication

- Two-step process to authenticate.
- At the webserver level.
- Secured by TLS.
- Username & Password are Base64 Encoded.
- Not the best choice for users.

HTTP Digest Authentication

- Multi-step process to authenticate.
- At the webserver level.
- Secured by TLS.
- Password is Hashed with nonce.
- Not the best choice for users.

Page Level Authentication

- Secured by TLS.
- Username & Password cleartext POST.
- Now you have to store passwords.

Password Storage

Cleartext Anyone?

Do NOT create your own method

Do NOT use MD5 (it is fast)

Password Storage

Two Key Points:

- Do use salt (long, unique, and random)
- Do use a cryptographically secure hash

SCrypt

A password-based key derivation function (password-based KDF) is generally designed to be computationally intensive, so that it takes a relatively long time to compute (say on the order of several hundred milliseconds). $\sim$ *Wikipedia*

SCrypt

Memory Hard

BCrypt

bcrypt is a password hashing function [...] incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power. ~ *Wikipedia*

BCrypt

Time Hard

PBKDF2

- Password-Based Key Derivation Function 2
- Part of RSA PKCS series
- Time Hard
- Iteration recomendation from OWASP:
  - 64k in 2013, doubled every two years (2,048,000 today)

Which One?

NIST has issued Special Publication SP 800-132 on the subject of storing hashed passwords.

Basically they recommend PBKDF2.

My favorite is SCrypt.

But any of these are acceptable.

As are some of the newer kids on the block (Argon2).

Not Your Problem

Maybe you can leverage someone else's auth.

- Facebook
- Google
- Twitter

Database Encryption

- MIT CryptDB
- MS SQL Always Encrypted
- Backups in Cleartext?

Cleartext Anyone?

- Migration to Encrypted

Migration

Options:

- Force Reset
- Upgrade/Update

Migration: Upgrade

1. Add legacy_password column (BOOL)
2. Upgrade existing hash (double-hash)
3. Handle legacy_password == TRUE

1. Authenticate

Migration: Upgrade

1. Add legacy_password column (BOOL)
2. Upgrade existing hash (double-hash)
3. Handle legacy_password == TRUE

1. Authenticate
2. Update with single-hash

Hashes

A hash function is any function that can be used to map data of arbitrary size to data of fixed size. $\sim$ Wikipedia

Crypto Hashes

> *A cryptographic hash function is a special class of hash function that has certain properties which make it suitable for use in cryptography.* $\sim$ *Wikipedia*

> *Cryptographic hash functions have many information-security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication.* $\sim$ *Wikipedia*

Important!

There are many applications,

not all crypto hashes are suitable

for all applications.

Crypto Hashes

- Do NOT use (for secure hashing)
  - MD2, MD4 & MD5
  - SHA1

Message Digest

Hash

Message Auth Code

> *[. . .] short piece of information used to authenticate a message [. . .] ∼ Wikipedia*

> *[. . .] protects both a message's data integrity as well as its authenticity [. . .] ∼ Wikipedia*

Key should be shared out-of-band.

Hash-based Message Auth Code

Basically, uses a cryptographic hash function to generate a MAC.

Compression

Correct Order of Operations:

1. Compress
2. Encrypt
3. MAC

Symmetric Encryption

Same key at both ends.

Asymmetric Encryption

Different keys at each end.

(Uses a key pair: Public and Private)

Symmetric Encryption

- RC4 is right out
- AES
- Not ECB mode of AES
- Key size of 128bits or greater
- CBC for block mode
- CTR for stream mode
- Random Initialization Vector
- Random Key
- Random Padding
- Never reuse Key and IV

Asymmetric Encryption

RSA with 2048 bit key or ECDSA/ECDH with 256 bit key

Pseudo-Random Numbers

Computers are not good at random.

Cryptographically Secure Pseudo-Random Numbers

Needed for things like the IV.

The End

References

- Fallible Inc's Security Guide for Developers
- Securing Your API Endpoints by Seth Petry-Johnson (Stir Trek 2016)
  - Securing Your API Endpoints - Seth Petry-Johnson - YouTube
  - Securing-Your-API-Endpoints - StirTrek-2016 - GitHub
- Common Developer Crypto Mistakes by Kevin W. Wall (Stir Trek 2016)
  - Common Developer Crypto Mistakes - Kevin W. Wall - YouTube
- You're Probably Storing Passwords Incorrectly
- ASIC/FPGA attacks on modern password hashes
- How to Safely Store Your Users' Passwords in 2016
- STRONGER KEY DERIVATION VIA SEQUENTIAL MEMORY-HARD FUNCTIONS by COLIN PERCIVAL
- CryptDB
- MS SQL Always Encrypted (Database Engine)
- Microsoft Description of Symmetric and Asymmetric Encryption
- Guide to Cryptography