Web Crypto for the Developer Who Has Better Things to Do
Or something like that...

OWASP AppSecAsiaPac 2012
Me, Myself and I.

- Adrian Hayes
- I'm a Kiwi
- Security Consultant at Security-Assessment.com
  - Penetration Tester
  - Source Code Reviewer
    - Java, .Net, Objective-C (evil apple), PHP, COBOL (god help me) etc...
  - Whatever else comes along
- Ex web app dev
  - Mainly JVM based stuff
What's This About?

Cryptography is the practice and study of hiding information

- We don't want people stealing our data
- But we do want some people to **Create**, **Read**, **Update** and **Delete** our data
- Smart cryptographers have given us the **concepts** to do this
- Smart programmers have given us the **tools** to do this
- Practical programmers have given us **nice tools** to do this

So let's use them.
Agenda

- Crypto Rules
- Random Token Generation
- Password Storage
- Backup Storage
- HTTPS
Crypto Rules

- Thou shalt not implement thy own low level crypto
- Thou shalt not reinvent thy crypto wheel
- Thou shalt be paranoid about thy crypto

- Thou shalt ensure thy web app is pentested by a reputable pentesting company...
Implementing cryptographic algorithms is like rolling naked down a hill.

Except that hill is made of tigers

Hungry, pissed off tigers
Tarsnap

*Online backups for the truly paranoid*

http://www.tarsnap.com/

- Implements PKI encrypted backups to the 'cloud'
- Works like *nix's tar utility, but way awesomer
- Implements it's own crypto...
A small code change meant an Integer was not incremented.
  
  - (nonce++ became just nonce)

Which ends up breaking the entire encryption scheme

- Damn
Crypto Rules

Don't Implement Your Own Crypto

There are lots of really good libraries out there

Lets use them
Random Token Generation

A string, that's random.
Simple right?

- Computers are really bad at random.
- Humans are also really bad at random.

This is not a good thing for security.
Random Token Generation

- Pseudo random
  - Something that looks random, but really isn't.
  - Often this is random enough. Unguessable is fine.

- General Token Generation Process
  - Grab some data that is unguessable (how?)
  - Use it to seed a strong pseudo RNG
  - Grab bytes from the generator and convert them to a string
Random Token Generation

Java

UUID.generateRandom().toString();

- 122bits of strong pseudo random goodness
- Which is $\approx 5.316911983 \times 10^{36}$ different possibilities
- Which is a lot

067e6132-3b6f-4be2-a171-2470e63dff20
Random Token Generation

Java

SecureRandom rand = new SecureRandom();
new BigInteger(128, rand).toString(32);

- 128 bits of randomness encoded in base32
- Change 128 to whatever length you require

25kkl0sn1rh3ec1o00p3oc6mvp
Random Token Generation

C# .NET

randBytes = new byte[16];
new RNGCryptoServiceProvider().GetBytes(randBytes);
Convert.toBase64String(randBytes);

- 128 bits of randomness encoded in base64
- Change byte[16] to whatever length you require

aEbAesx5FKxzX0FXLQp5Yw==
Random Token Generation

**PHP**

```php
base64_encode(openssl_random_pseudo_bytes(16))
```

- 128 bits of randomness encoded in base64
- Change 16 to whatever
- PHP 5.3.0 with openssl module
- Can be slow on Windows

D8fZLgyBy8t0M1KXjTS8gg==
Random Token Generation

Ruby

```ruby
require 'active_support/secure_random'
random_string = ActiveSupport::SecureRandom.hex(16)
```

- 128 bits of randomness encoded in hex
- Change 16 to whatever
- Requires ActiveSupport

a5163bef582fccad88dd03f98815e001
Password Storage

- Lots of web apps get it wrong
- Most of web apps don't get it right

- Concepts
  - Hashes
  - Salts
  - Speed

“We call this one the 'Password Manager.' The vest is made of Post-It notes.”
Password Storage

- Yeah but, who cares?
  - Me, the people using your app, your boss when you get hacked, your shareholders, the media, hackers, probably a bunch of other people and me again.

- Sony hacked by Lulzsec – June 2011
  - 51,000 account credentials stolen
  - Passwords stored in clear text

- Rockyou.com – December 2009
  - 32 million account credentials stolen
  - Passwords stored in clear text
Password Storage

- We need to passwords to identify people
  - We ensure the password they provide on login is the same as the password they entered on registration.
  - We have to allow people to change and reset their password.

- None of this requires we store the actual password.
  - We can just store it's cryptographic hash.
A cryptographic hash takes bytes as input, and provides a fixed length byte output.

- A good hash is (according to wikipedia)
  - Easy to compute
  - Infeasible to reverse
  - Infeasible to create a “collision”

- Lots of well known hashing algorithms
  - MD5, SHA-1, NTLM, RIPEMD, WHIRLPOOL etc
Password Storage

Easy to compute?
Seriously?

- We crack secure hashes by trying possible inputs until one matches.
- We can now generate billions of MD5 password hashes per second using off the shelf GPUs.

This is not good.
Password Storage

- For passwords we need:
  - A hash that is unavoidably **Slow**.
  - A hash that is **Long**
  - **Salts** to make it taste better (and defeat rainbow tables)

So what does that?

**bCrypt**

Yay!
Password Storage

• Why bCrypt?
  • bCrypt is configurably slow
  • bCrypt handles salts for us
  • bCrypt has been ported to most languages

It's really just a nice solution
Password Storage

Creating a Hash

(Registration and password change/reset)

BCrypt.hashpw("myPass", Bcrypt.gensalt(10));

• Generates a salt + hash in one nice string
• Using a “work factor” of 10
Password Storage

Checking a Hash
(On login and password change)

BCrypt.checkpw("myPass", hashFromDB);

- Uses salt from hash in DB
- Rehashes password and checks for match
Password Storage

Advanced Technique

- Use an Application Specific Salt
  - Use bCrypt as normal but include another salt as well
  - Can't crack hashes unless I own both the DB and the Application
  - Remember bCrypt only uses the first 72 bytes of a password. So 15 character salt must come last.

```java
String APP_SALT = "0I)5w9Zi$hbdi7S";
Bcrypt.hashpw("myPass"+APP_SALT, BCrypt.gensalt(10));
```
Password Storage

Java

C# .NET
- http://bcrypt.codeplex.com/

PHP
- http://www.openwall.com/phpass/

Ruby
- http://bcrypt-ruby.rubyforge.org/

Python
- http://code.google.com/p/py-bcrypt/
Backup Storage

- Backups are a gold mine and often not protected
  - Database info
  - Passwords
  - Source code

- Concept
  - Public Key Encryption
Backup Storage

- Your web app needs to be backed up
- But generally doesn't need to manage the backups

- So how do we store backups safely?
  - They should be writeable
  - But not deleteable or updateable
  - And not readable by the application
So... What's this Public Key Crypto Stuff?

- **Public Key Crypto** (or asymmetric crypto)
  - Two keys, a public one, a private one
    - Public is used for encryption,
      - Public cannot decrypt your backups
  - Private is stored somewhere safe (like in a safe)
    - Private can decrypt backups
    - Private is for testing and emergencies only
Backup Storage

- Backups are encrypted with the public key
  - Written somewhere safe
  - The app can only write, not update or delete

- Restoration is performed manually
  - Private key is required and grabbed from the safe
distribute.IT
June, 2011

Got hacked
Backups not protected
4800 hosted sites gone

distribute.IT no longer exists

Damn
Introducing GnuPG

- Provides secure public key encryption
- Easy to use
- Can't really go wrong with it (providing you're not an idiot)
Backup Storage

1. Generate your keys
2. Export your keys
3. Delete key from local keyring
4. Import your public keys to the server doing backups
5. Store your private keys in a SAFE place

- Do your restore tests regularly. Seriously.

Seriously. Restore Tests.
Backup Storage

- **Create a keypair** *(defaults are good)*
  
  `gpg --gen-key`

- **List current keys**
  
  - `gpg --list-keys`
  
  - `gpg --list-secret-keys`

- **Export Keys**

  - `gpg --export --armor <keyId>`
  
  - `gpg --export-secret-keys --armor <keyId>`

- **Delete Keys**

  - `gpg -delete-secret-and-public-key <keyId>`
Backup Storage

- Encrypt a File
  
  `gpg --encrypt -r <keyId> <filename>`

- Decrypt a File
  
  - `gpg --decrypt <filename>.gpg`

Pretty Simple
HTTPS

HTTPS means SSL/TLS

Which means point to point client/server encryption

Generally

- Concepts
  - Versions and Ciphers
  - Man in the Middle attacks
HTTPS should be used anywhere sensitive information is passed to or from a web app

- Passwords
- Auth tokens (firesheep)
- Credit cards (pci dss anyone?)
- HTML assets on a HTTPS page
  - JavaScript
  - CSS
  - Images
HTTPS

You just turn it on right?

Almost.

Some web servers have insane defaults.
SSL/TLS Versions and Ciphers

- Ciphers consists of
  - Public Key Encryption type
  - Symmetric Key Encryption type
  - Block Mode of Operation
  - Digest Algorithm
- Such a thing as NULL ciphers
- SSLv2 is *broken as f**k*, don't use it
- TLS had a renegotiation bug, must be patched
- CBC Mode vulnerable to the BEAST attack
HTTPS

Way too complicated.

Let's use a tool to help us

https://www.ssllabs.com/ssldb/index.html
SSL Report: www.google.com (74.125.45.104)
Assessed on: Tue Jul 05 18:12:54 UTC 2011 | Clear cache

Summary

Overall Rating

Certificate: 100
Protocol Support: 85
Key Exchange: 80
Cipher Strength: 90

85

The scores are explained in the SSL Server Rating Guide 2009.

This server supports secure renegotiation

ssl labs.com
SSL Report: [redacted]

Summary

Overall Rating

Certificate: 100
Protocol Support: 85
Key Exchange: 40
Cipher Strength: 60

The scores are explained in the SSL Server Rating Guide 2009.

This server is vulnerable to MITM attacks because it supports renegotiation (more info here).

ssl labs.com
Man in the Middle Attacks

HTTPS protects against these right?

Kind of.

Heard of SSLStrip?
HTTPS

SSLStrip

- Intercepts HTTPS
  - Rewrites HTTPS links to HTTP
- Victim connects through SSLStrip proxy via HTTP
- SSLStrip connects to server via HTTPS
- Everything looks fine to both server and victim!
HTTPS

SSLStrip

User's Browser

Evil Blackhat Man in the Middling Hacker. Who is evil.

MyBank HTTP Server

http://mybank.com

https://mybank.com
So, what do we do?

Google to the rescue with Strict Transport Security Header

- **Header**: `Strict-Transport-Security: max-age=2592000`
- HTTPS will be forced for 30 days
- Supported by Chrome and Firefox *(it's a start)*
- User must have visited the site before
Finally

So there you have it.

Questions?

Email: adrian.hayes@security-assessment.com