WEB SECURITY

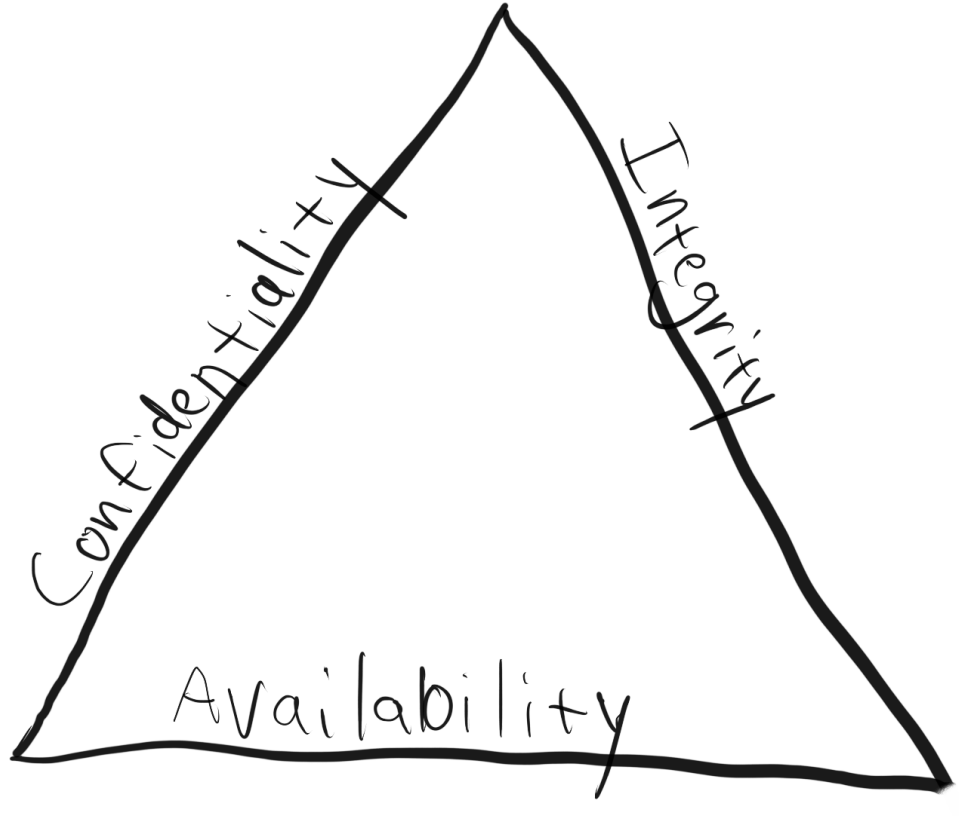
THE PHILOSOPHY
THE VULNERABILITIES
THE DEFENSES

> *A fool may ask more questions in an hour than a wise man can answer in seven years.*
>
> — English proverb

> *A ~~fool~~ lunch-and-learn may ask more ~~questions~~ security improvements in an hour than a ~~wise man~~ developer can answer in seven years.*
>
> — Me

THE PHILOSOPHY

1. SECURITY HAS THREE DIMENSIONS.



2. DON'T CONFUSE VULNERABILITIES, THREATS, AND RISKS.

Threat – “anything (e.g., object, substance, human, etc.) that is capable of acting against an asset in a manner that can result in harm”

Vulnerability – “always dependent upon the type and level of force being applied”

Risk – “the probable frequency and probable magnitude of future loss”

— Jack Jones, “Bald Tire”

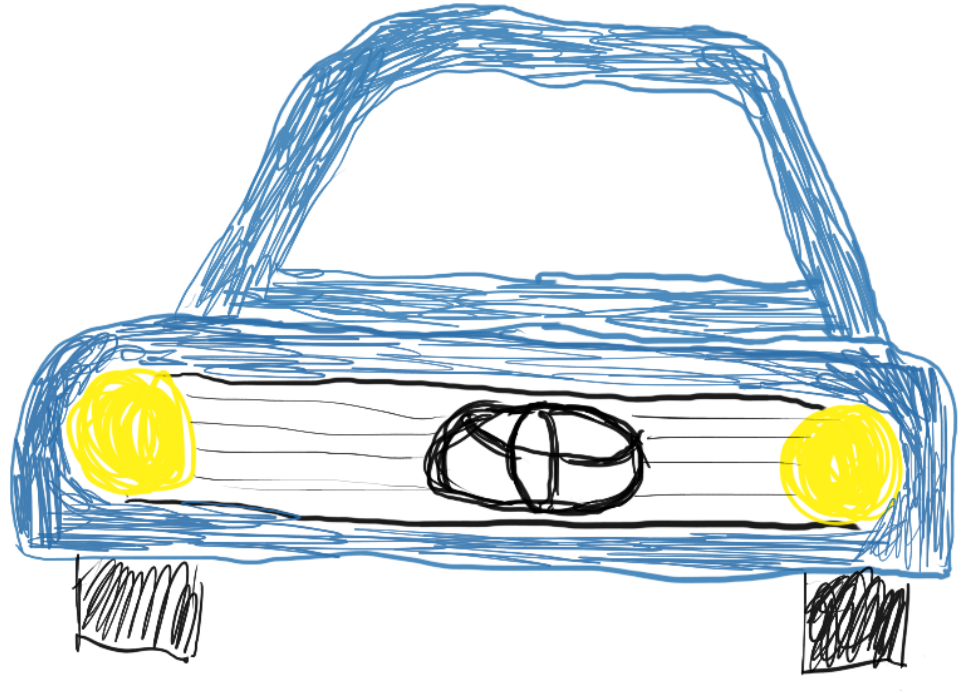
3. SECURITY IS ASYMMETRICAL.

4. MURPHY WAS AN INCURABLE OPTIMIST.

> *Our task is to program a computer which gives answers which are subtly and maliciously wrong at the most inconvenient possible moment.*

— Ross Anderson and Roger Needham, “Programming Satan’s Computer”

5. I MAKE MISTAKES.



Outputting arbitrary string values in PHP

```
<p>Hello, <?php echo $first_name . ' ' . $last_name; ?>!</p>
```

Safely outputting string values in PHP

```
<p>
    Hello,
    <?php echo htmlspecialchars($first_name)
        . ' ' . htmlspecialchars($last_name); ?>!
</p>
```

Safely outputting string values in Django

```
<p>Hello, {{first_name}} {{last_name}}!</p>
```

Outputting arbitrary string values in Django

```
<p>Hello, {{first_name|safe}} {{last_name|safe}}!</p>
```

Safely outputting string values in React

```
<p>Hello, {first_name} {last_name}!</p>
```

Outputting arbitrary string values in React

```
<p>
    Hello,
    <span dangerouslySetInnerHTML={{
        __html: first_name + ' ' + last_name
    }}/>!
</p>
```

THE VULNERABILITIES

SQL INJECTION

- Sanitize inputs.
- Use parameterized queries.

```
query("SELECT * FROM users WHERE username = '" + username + "'
```

```
query('SELECT * FROM users WHERE username = ?;', username);
```

LDAP INJECTION, COMMAND-LINE INJECTION, ETC.

```
system("rm -rf $path")
```

```
subprocess.run(['rm', '-rf', path])
```

XSS

- Three types: reflected, persistent, DOM-based
- Escape your outputs.
- Sanitize inputs using a whitelist approach.

CSRF

- Use a CSRF token.
- Consider using referrer checking.

CLICKJACKING

Use a Content Security Policy.

Use X-Frame-Options.

Use a frame-breaking script.

THE DEFENSES

THE BASICS

- Never trust the user.
- Sanitize all inputs.
- Validate everything.
- Use secure-by-default approaches.
- Use a secure password hash.
- Use a reasonable password policy.

HTTPS

Do it for everything.

```
<script src="//ajax.googleapis.com/sample.js">
```

Let's Encrypt, Cloudflare

HTTPS

Do it correctly

SSLv2 (deprecated 2011), SSLv3 (deprecated 2015),
TLSv1.2, TLSv1.3 (released 2018)

SSL Labs

SECURE COOKIES

HttpOnly - no JavaScript

```
Set-Cookie: MyCookieName=value; path=/; HttpOnly
```

```
<httpCookies httpOnlyCookies="true" />
```

Secure - no plaintext HTTP

```
Set-Cookie: MyCookieName=value; path=/; secure
```

```
<httpCookies requireSSL="true" />
```

SAMESITE COOKIES

`strict` - never submitted when entering from an external link

`lax` - not submitted when POST'ed externally

Lax is still better than the default!

https://www.owasp.org/index.php/SameSite

STRICT TRANSPORT SECURITY (HSTS)

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Chrome Dev Tools show HTTP 307 - "Internal Redirect."

HSTS Preload

SUBRESOURCE INTEGRITY (SRI)

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
    integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5sm
    crossorigin="anonymous">
```

SRI - GETTING HASHES

3 approaches:

Fake a hash then read Chrome Dev Tools.

```
openssl dgst -sha384 -binary jquery.min.js |
    openssl base64 -A
```

https://www.srihash.org/

SRI - PROVIDING A FALLBACK

```
<script>
(window.jQuery) ||
    document.write('<script src="/scripts/jquery"></script>');
</script>
```

PUBLIC KEY PINNING (HPKP)

```
Public-Key-Pins:
    pin-sha256="cUPcTAZWKaASuYWhhneDttWpY3oBAkE3h2+soZS7sWs=";
    pin-sha256="M8HztCzM3elUxkcjR2S5P4hhyBNf6lHkmjAHKhpGPWE=";
    max-age=5184000; includeSubDomains;
    report-uri="https://www.example.org/hpkp-report"
```

CERTIFICATE AUTHORITY AUTHORIZATION (CAA)

```
example.com.   IN   CAA 0 issue "ca.example.net"
example.com.   IN   CAA 0 iodef "mailto:security@example.com"
example.com.   IN   CAA 0 iodef "http://iodef.example.com/"
```

CONTENT SECURITY POLICIES (CSP)

```
Content-Security-Policy: default-src 'self';
    img-src *; media-src media1.com media2.com;
    script-src userscripts.example.com
```

CONTENT SECURITY POLICIES (CSP)

Using a nonce for inline scripts:

```
Content-Security-Policy: default-src 'self';
    script-src 'nonce-4AEemGb0xJptoIGFP3Nd'
<script type="text/javascript" nonce="4AEemGb0xJptoIGFP3Nd">
```

CONTENT SECURITY POLICIES (CSP)

My Blog Now Has a Content Security Policy - Here's How I've Done It, Troy Hunt

CONTENT SECURITY POLICIES (CSP)

upgrade-insecure-requests

BUT WAIT, THERE'S MORE!

- Log errors
- Hide errors
- Reduce information leakage (like server versions in response headers)

BUT WAIT, THERE'S MORE!

- Credentials stuffing
- Secure password reset
- Account enumeration via email address submission
- 2-factor authentication
- Make it somebody else's problem

BUT WAIT, THERE'S MORE!

OWASP

Troy's Ultimate List of Security Links

securityheaders.com

REFERENCES AND CREDITS

- Troy Hunt
- Bald Tire, Jack Jones
- BSD daemon artwork
- Programming Satan's Computer, Ross Anderson and Roger Needham
- Strict-Transport-Security , MDN
- HTTP Public Key Pinning (HPKP) , MDN
- DNS Certificate Authority Authorization , Wikipedia
- CSP , MDN
- Locking Down Your Website Scripts with CSP, Hashes, Nonces and Report UI , Troy Hunt