KF5002 Web Programming
Security in practice

The Royal Academy of Engineering funded a Visiting Professorship in Practical Cybersecurity Insights at Northumbria University, 2019–2022. These slides are a slightly modified version of those delivered, intended to be available after the project has ended

Web `https://green-pike.co.uk/nvp`
Email p.brooke@northumbria.ac.uk (until it stops working. . .)
phil@green-pike.co.uk

Why. . . ?

. . . are you building a web app?

Consider

- Intended / possible users
- What you intend them to do
- What features you're making available
- Who might attack or abuse it

Scanning

Malicious scanning is routine and pervasive.
e.g., Suricata logs:

```
Signature_ID=2010935, count=40
      1 101.254.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 111.40.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 112.44.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 113.10.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 117.149.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 117.34.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 121.52.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 122.228.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 125.73.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      1 12.91.nn.nn,null,ET SCAN Suspicious inbound to MSSQL port 1433
      ...
```

And more!

```
Signature_ID=2006402, count=1
      1 24.28.nn.nn,null,ET POLICY Incoming Basic Auth Base64 HTTP Password detected unencrypted

Signature_ID=2010937, count=4
      1 139.162.nn.nn,null,ET SCAN Suspicious inbound to mySQL port 3306
      3 185.216.nn.nn,null,ET SCAN Suspicious inbound to mySQL port 3306

Signature_ID=2010939, count=3
      2 217.21.nn.nn,null,ET SCAN Suspicious inbound to PostgreSQL port 5432
      1 240e:00f7:nnnn:nnnn:0000:0000:0000:0003,null,ET SCAN Suspicious inbound to PostgreSQL port 5432

Signature_ID=2019526, count=1
      1 124.40.nn.nn,null,ET WEB_SERVER WEB-PHP phpinfo access

Signature_ID=2022268, count=1
      1 124.40.nn.nn,null,ET EXPLOIT Joomla RCE M3 (Serialized PHP in XFF)
```

Database considerations

Attackers would *really* like your database

So make sure it's unreachable from outside

- Small-scale / single-host systems — ensure DB ports are closed
- Larger / multi-host systems — DB servers should allow access from only the middle-layer servers
- Strong credentials for access to the DB

Classic host / web server security

- Close all unneeded ports
- Check that you closed all unneeded ports...
- Use HTTPS
  - Use a good certificate authority (no self-signed certificates!)
- Redirect HTTP to HTTPS (or reject HTTP)
- For some types of processing, consider refusing downgrade of SSL cipher suites, *i.e.*, *require* a good standard to access the system at all

HTTP headers

Many, many standards...

- HTTP Strict Transport Security (HSTS): *e.g.*,

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

  Directs clients to use HTTPS; if HTTPS is unavailable, to refuse the connection

- Refuse/control framing by other sites (helps defeat click-jacking)

```
X-Frame-Options: DENY
```

  Arguably obsolete, with

```
Content-Security-Policy: frame-ancestors 'none'
```

  an alternative

HTTP headers

- Content-Security-Policy is powerful. Example from a web app that takes payments and uses some Google fonts

```
Content-Security-Policy: default-src 'self' https://checkout.stripe.com https://js.stripe.com;
                         img-src 'self' https://*.stripe.com;
                         style-src 'self' https://fonts.googleapis.com;
                         font-src 'self' https://fonts.gstatic.com
```

More database thoughts

Monitor for exfiltration:

- Should the database server be making outbound SSH or HTTP requests?
- Outbound firewalls?

Avoid storing credentials on servers at all (can be messy)

Backup regimes — important for business continuity as well as in case of attack

Impacts of database compromise

(Assuming you know you've been compromised...)

- GDPR reporting obligations (end-users, data subjects, ICO)
- Reputational harm, loss of income, ...
- Cost of remediation, rebuild

Impacts of database compromise — passwords

You've been told to not store plaintext passwords (store salted hashed passwords instead)

Why?

Impacts of database compromise — passwords

You've been told to not store plaintext passwords (store salted hashed passwords instead)

- (Obvious) discloses passwords for the compromised system
- *Internal malfeasors* could abuse the passwords
- *Users often re-use passwords*
  $\Rightarrow$ Means we might need users to reset passwords across *many* systems
- Databases of compromised usernames / emails and passwords are available via multiple "black" forums/sites, including from "old" breaches

Impacts of database compromise — passwords

Examples:

- https://nakedsecurity.sophos.com/2013/11/20/serious-security-how-to-store-your-users-passwords-safely/
- https://cloud.google.com/blog/products/g-suite/notifying-administrators-about-unhashed-password-storage
- https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-f

User management

Web apps usually involve users...
So now you need to manage users

- End-user creation
  - Validation by emailing a token?
- Existing database or federation — LDAP, Active Directory, OAuth, SAML
- Social media login, *e.g.*, Facebook Login
- Duplicate users
- Password expiry
- Account expiry

Validation

Your tutors have already told you about the need for *validation* and *sanitisation*.

- Prevent database (SQL) injection
- Prevent tampering / injection into later page generation (*e.g.*, to enable cross-site request forgery, pop-ups)
- (XKCD Bobby Tables)

Where to validate?

1. Should you validate input on the client-side? (Why?)
2. Should you validate input on the server-side? (Why? What if you already validate on client-side?)

Validation and character sets

Are you using ASCII? UTF-8? UTF-16?

What is sent to your app?

OWASP (appendix D of the testing guide):

- "For instance, a / can be represented as 2F (hex) in ASCII, while the same character (/) is encoded as C0 AF in Unicode (2 byte sequence)."
- "Multibyte encoding has been used in the past to bypass standard input validation functions and carry out cross site scripting and SQL injection attacks."
- Misuse of numeric encodings (*e.g.*, hex)

Validation and character sets

Many programming languages are not directly aware of multibyte encoding

- C strings are simply a sequence of octets, with the variable holding the starting memory address

PHP has a set of multibyte string functions:
`https://www.php.net/manual/en/ref.mbstring.php`

> *"Multibyte character encoding schemes and their related issues are fairly complicated"*

Validation and character sets

Suggestions for further reading:

- The Open Web Application Security Project (OWASP)
  `https://www.owasp.org/` is an excellent resource
- Internationalized Domain Names in Applications (IDNA)
- Punycode

Aside: As well as allowing *many* more ASCII domain names, ICANN also allows UTF-8 domain names... beware of homograph attacks *a.k.a.* script spoofing

Session handling

Classic mistake:

- Assume that only one user is using your app at a time

Temporary file / temporary tables

Inadvertent re-use and/or resource consumption

XHR and AJAX

Using XMLHttpRequest (XHR) calls instead of generating complete HTML pages for each interaction is a common model for web apps

- Supports AJAX ("Asynchronous JavaScript and XML")
- Often a better end-user "experience"
- Typically uses JSON (not XML)

All the previous comments and advice about databases, sanitisation, validation, *etc.* apply to these interfaces!

. . . and to other "stacks" (*i.e.*, regardless of web server, language(s), database)

“Approval”

How to convince an auditor the system is secure enough?

- Clear layers of internal APIs, *e.g.*,
  - “outside” which always cleans
  - “inside” which is unreachable otherwise
- Clear documentation and code
- Systematic, repeatable testing
  - Unit testing
  - Automatic test frameworks
  - . . . but can be challenging with UIs and heavy use of state
- Fuzzing — always test for unexpected usage
- Scanners — *e.g.*, Nessus (commercial) or OpenVAS

Things to try

- Injection attacks
- Password cracking (*e.g.*, hashcat and md5)
- Salting
- OpenVAS

The end

Web	`https://green-pike.co.uk/nvp`
Email	p.brooke@northumbria.ac.uk (until it stops working. . .)
	phil@green-pike.co.uk