Getting Inside Common Web Security Threats

ANDY LONGSHAW AND EOIN WOODS

SPA2015, JUNE 2015

Introductions

- Andy Longshaw
  - Solution Architect at Advanced Legal
  - Inveterate worrier about system qualities like availability and… security
  - Responsible for a public-facing, cloud-based web system
- Eoin Woods
  - CTO at Endava
  - Long time security dabbler
  - Increasingly concerned at level of cyber threat for "normal" systems

Goals

- Introduce OWASP and the Top 10 Vulnerabilities List
- Illustrate some of the Top 10 by exploiting them ourselves
- Show how real attacks combine vulnerabilities
- Introduce some useful tools
  - Mutillidae, BurpSuite, SQLMap

Content

- The OWASP Top 10
- The Tools We'll Use
- Exercises
- Reviewing Defences
- Summary

The OWASP Top 10

OWASP

- The Open Web Application Security Project
  - Largely volunteer organisation, largely online
- Exists to improve the state of software security
  - Performs research
  - Develops tools
  - Publishes guidance and informal standards
  - Runs local chapters for face to face meetings (a dozen in the UK alone)
- "OWASP Top 10" project lists the top application security risks
  - Referenced widely by MITRE, PCI DSS and similar
  - Updated every few years (2003, 2004, 2007, 2010, 2013)

OWASP Top 10

- #1 Injection Attacks
- #2 Authentication and Session Management
- #3 Cross Site Scripting (XSS)
- #4 Direct Object Reference
- #5 Security Misconfiguration
- #6 Sensitive Data Exposure
- #7 Function Level Access Control
- #8 Cross Site Request Forgery (CSRF)
- #9 Component Vulnerabilities
- #10 Unvalidated Redirects and Forwards

These may look "obvious" but appear on the list year after year, based on real vulnerability databases!

#1 Injection Attacks

- Unvalidated input passed to an interpreter
  - Operating system and SQL are most common

```
SELECT * from table1 where name = '%1'
```

Set '%1' to ' OR 1=1 --

Result => `SELECT * FROM table1 WHERE name = '' OR 1=1 --`

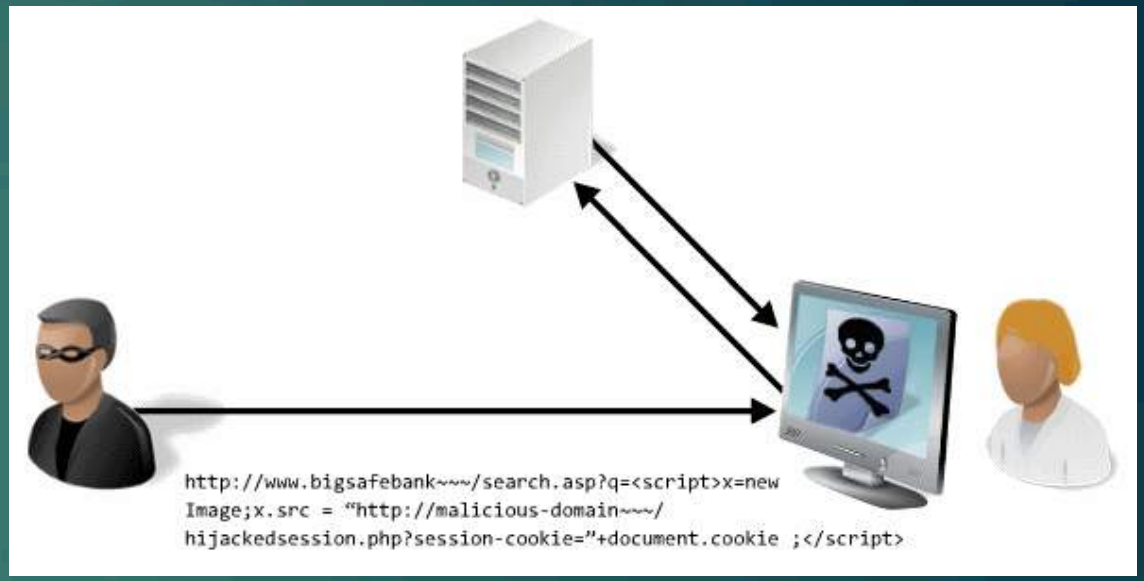
- Defences include "escaping" inputs, using bind variables, using white lists, …

#2 Broken Authentication or Session Management

- HTTP is stateless => some sort of credential sent every time
- Credential sent over non-TLS connection can be tampered with
- Session ID often displayed yet often as good as login details
- Defences based on strong authentication and session management controls

#3 Cross Site Scripting

- Slightly misleading name – occurs any time script is injected into a user's web page
  - Reflected attack – crafted link in email, on a forum, …
  - Persistent attack - database records, site's postings, activity listings
- Allows redirection, session data stealing, page corruption, …
- Defences include validation and escaping on the server-side



#4 Insecure Direct Object Refs

- Directly referencing filenames, object IDs and similar in requests
- Not authenticating access to each on the server
  - e.g. relying on limited list of options returned to client
- Allows client to modify request and gain access to other objects

```
http://mysite.com/view?id=file1.txt
```

… how about `http://mysite.com/view?id=../robots.txt` ??

- Defences include using pseudo references on client and authenticating all object accesses

#5 Security Misconfiguration

- Security configuration is often complicated
  - Many different places to put it, complex semantics
  - Layers from OS up to application all need to be consistent
- It is easy to accidentally miss an important part
  - OS file permissions?
  - .htaccess files?
  - Shared credentials in test and production?
- Allows accidental access to resources or even site modification

- Mitigation via scanning, standardisation, simplicity and automation

#6 Sensitive Data Exposure

- Is sensitive data secured in transit?
  - TLS, message encryption
- Is sensitive data secured at rest?
  - Encryption, tokenisation, separation
- Loss of data (e.g. credit card numbers) or spoofing attacks
- Mitigation via threat analysis, limiting scope of data, standardisation

#7 Function Level Access Control

- Relying on information sent to the client for access control
  - e.g. page menu omitting “update” and “delete” option for a record
  - Not checking the action (function) being performed on the server
- Client can guess or infer the right request form for the other actions
  - Bypassed security model - also see #4 Insecure Object References

`http://www.example.com/gettxn?txnid=4567`

→ `http://www.example.com/updttxn?txnid=4567&value=1000.00`

- Never trust the client - check authorisation for every request

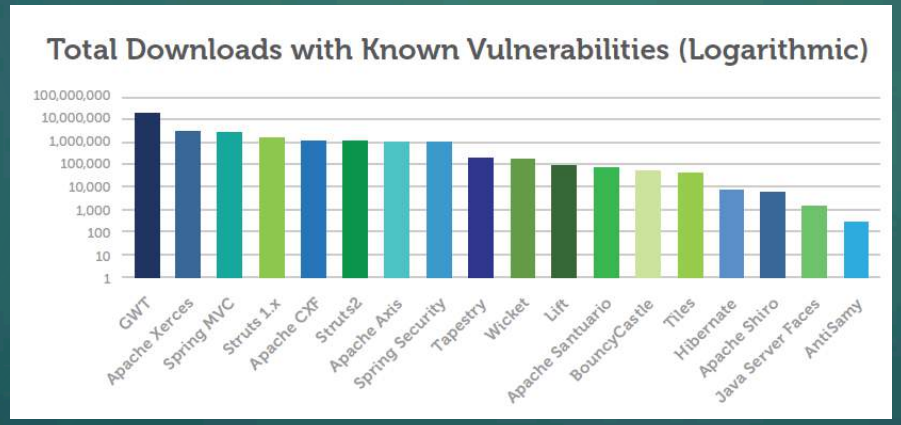
#8 Cross Site Request Forgery

- User triggers malicious code that submits fraudulent request using browser security context
  - e.g. clicking a link => run JavaScript => change Github password
- Various subtle variations on this make defence quite difficult
  - How you do you know it is the user?

- Primary defence is the “challenge value” in pages
  - Expect the challenge value from the latest page in any request
  - More authentication steps for sensitive operations
  - Short sessions with real logout process

#9 Known Vulnerable Components

- Many commonly used components have vulnerabilities
  - See weekly US-CERT list for a frightening reality check!
  - Many open source libraries don't have well researched vulnerabilities
- Few teams consider the security of their 3rd party components
  - And keeping everything up to date is disruptive



- Consider automated scanning of 3rd party components, actively review vulnerability lists, keep components patched

#10 Unvalidated Redirects and Forwards

- Redirecting or forwarding to targets based on parameters

```
http://www.mysite.com/selectpage?pageid=emea_home.html

    -> http://www.mysite.com/selectpage?pageid=pishinghome.com
```

(Without careful validation this redirects user to malicious page)

- Avoid using parameters where redirect or forward is needed. Where parameter *is* needed use a key and map to URL on server

Summary of Attack Vector Types

- **Interpreter injections** – OS, SQL, …
- **Page injections** – HTML, XSS (JavaScript)
- **Lack of Validation** – trusting client side restrictions, allowing session IDs and cookies to be reused, not checking input fields thoroughly, using parameter values directly in pages and links
- **Not protecting valuable data** – data loss, spoofing, man in the middle, …
- **Underlying Platform** – configuration mistakes, vulnerabilities, complexity

Tools We'll Use

Mutillidae

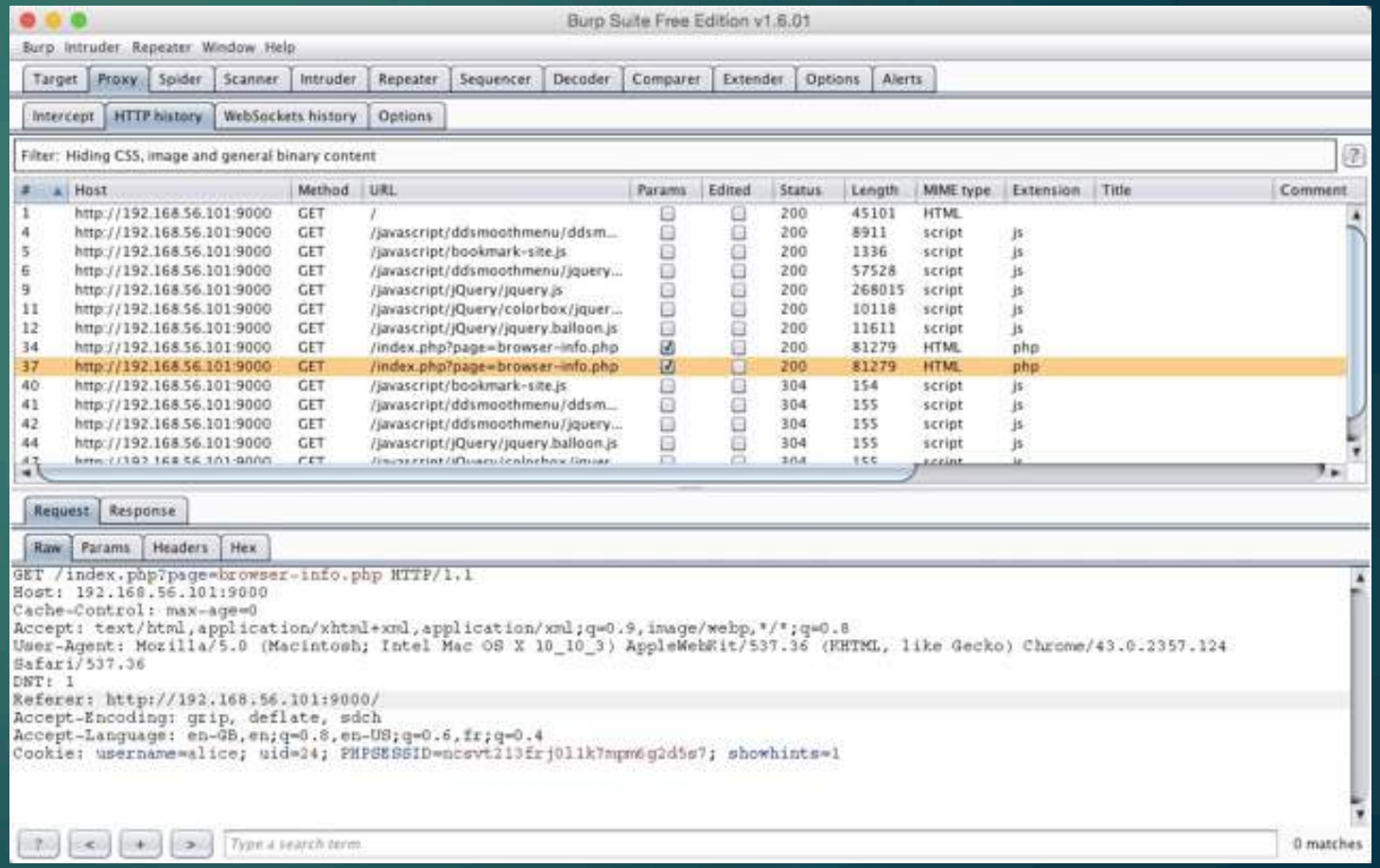
www.irongeek.com
http://sourceforge.net/projects/mutillidae/

- Deliberately insecure LAMP web application
- We have provided it in a VirtualBox VM
- Provides examples of the OWASP Top 10 in action
- We will use it to illustrate exploiting the vulnerabilities

BurpSuite

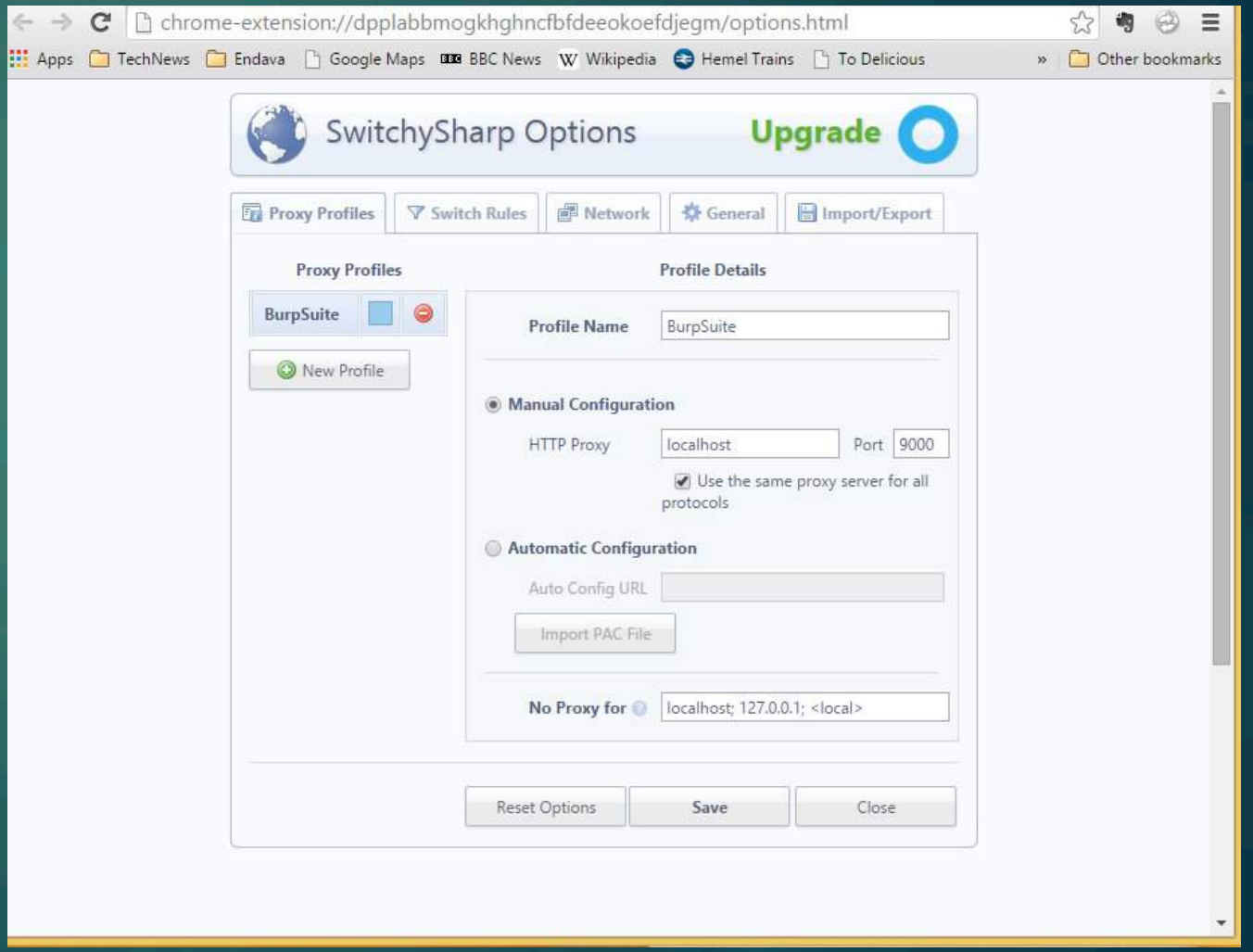
`http://portswigger.net/burp`

- Commercial proxy, scanning, pentest tool
- Very capable free version available
- Inspect traffic, manipulate headers and content, …
- Made in Knutsford!



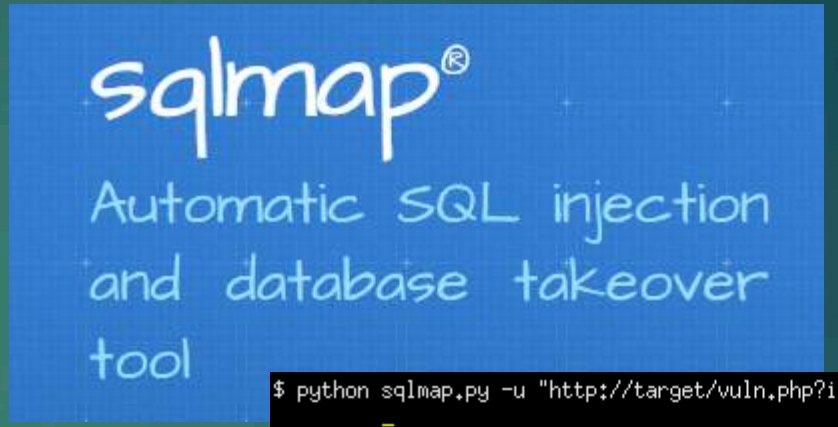
Browser and Proxy Switcher

- Chrome and SwitchySharp or other similar pairing
- Allows easy switching of proxy server to BurpSuite



SQLMap (optional)

- Automated SQL injection and database pentest tool
- Open source Python based command line tool
- Frighteningly effective!



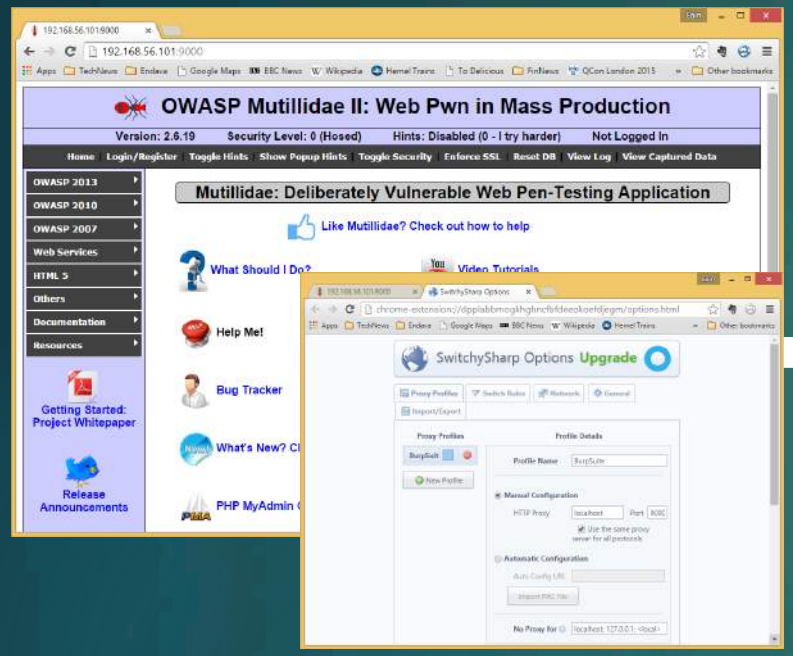

```
$ python sqlmap.py -u "http://target/vuln.php?id=1" --batch
                         _
 ___ ___ _| |_____ ___ ___  {1.0-dev-4512258}
|_ -| . | | |     | .'| . |
|___|_  |_|_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 15:02:07

[15:02:07] [INFO] testing connection to the target URL
[15:02:07] [INFO] heuristics detected web page charset 'ascii'
[15:02:07] [INFO] testing if the target URL is stable. This can take a couple of
 seconds
[15:02:08] [INFO] target URL is stable
[15:02:08] [INFO] testing if GET parameter 'id' is dynamic
[15:02:08] [INFO] confirming that GET parameter 'id' is dynamic
[15:02:08] [INFO] GET parameter 'id' is dynamic
[15:02:08] [INFO] heuristic (basic) test shows that GET parameter 'id' might be
injectable (possible DBMS: 'MySQL')
```

Exercises

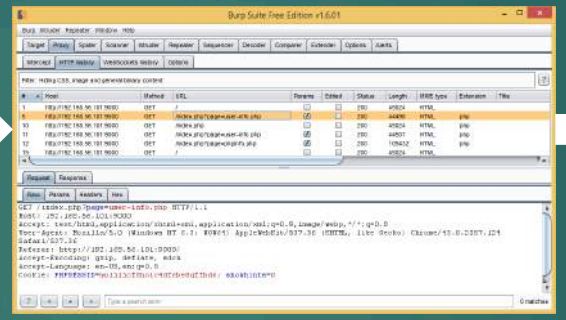
Structure of the Exercises

- **Scout out the system**
  - SQL injection attacks
  - Insecure direct object reference attack for a file
- **Get access to the operating system**
  - OS injection attack
  - Unvalidated file upload attack and inject PHP file into the web site
- **Get access to a user's account**
  - Write a blog post on behalf of someone else (session token attack)
- **Steal login credentials**
  - XSS attack using a crafted HTML form, JavaScript and a blog post

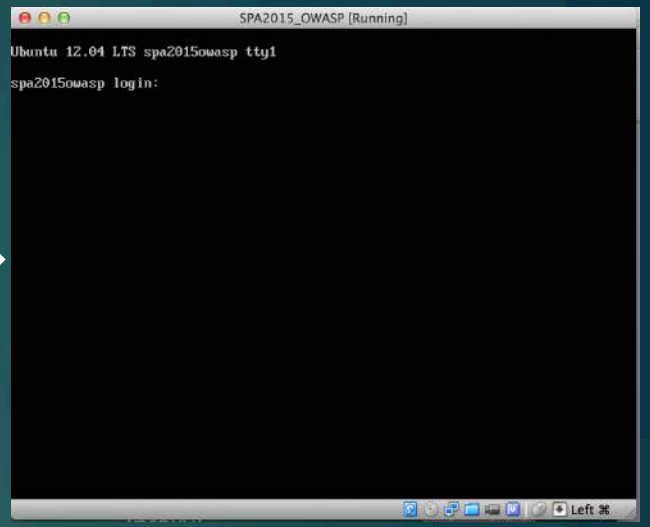
Getting Started



Browser with proxy plugin



BurpSuite (proxy)



Mutillidae

- Start Mutillidae in a VirtualBox VM
- Start BurpSuite and enable the proxy
- Configure browser to use BurpSuite proxy (localhost:9000)

Working with the Exercises

- Self paced exercises by yourself or in pairs
  - Self contained on your machine
- We provide:
  - Overview and instructions
  - Solutions if you want them
- As you go, reflect on what you're learning – we'll share at the end

Demonstrations

- SQL injection to list all users
- BurpSuite request interception
- JavaScript alertbox injection

Exercises

- 35 minutes setup and initial exercise
- 15 minute break
- 45 minutes further exercises

- Mutillidae URL
  - http://YOUR-VM-IP-ADDRESS/mutillidae

Reviewing Defences

Key Web Vulnerability Defences

- Don't trust clients (browsers)
  - Validate inputs, confirm authorisations, validate object references, …
- Identify "interpreters", escape their inputs, use bind variables, …
  - Operating system execution, SQL queries, JavaScript, …
  - Web page dynamic content (escape, validate, placeholders)
- Protect valuable information at rest and in transit
- Simplicity
  - Verify configuration and correctness
- Standardise and Automate
  - Force consistency, avoid configuration errors

Summary

Summary

- Much of the technology we use is inherently insecure
  - Mitigation needs to be part of application development
- Attacking systems is becoming industrialised
  - Digital transformation is providing more valuable, less secure targets
- Fundamental attack vectors appear again and again
  - Injection, interception, web page manipulation, missing validation, poor configuration, …
- Most real attacks exploit a series of vulnerabilities
  - Each vulnerability may not look serious, the combination is
- Most mitigations are not difficult but need to be applied consistently
  - … and may conflict with other desirable qualities

Andy Longshaw
andy@blueskyline.com
www.blueskyline.com
@andylongshaw

Eoin Woods
eoin.woods@artechra.com
www.artechra.com
@eoinwoods