COS80013 Internet Security

Week 8

Presented by Dr Rory Coulter

28 April 2025

SWINBURNE UNIVERSITY OF TECHNOLOGY

Week 8 Class

Assignment 2

3 portions, technical in nature

Weekly demonstrations will be given to help

- All files and artefacts are uploaded
- Additional labs have been provided for practice
- 2 June, 23:59
- Backup your work

Tools for Assignment 2

SIFT Workstation, Volatility, evtxexport, strings

Weekly demonstrations will be given to help

- Approach to a forensic workstation
- Install of tools
- https://github.com/volatilityfoundation/volatility/wiki/installation
- https://letsdefend.io/blog/how-to-install-volatility-2-and-volatility-3-on-linux
- https://github.com/volatilityfoundation/volatility/wiki/Command-Reference
- https://github.com/libyal/libevtx/tree/main

COS80013 Internet Security

Lecture Week 8A

SWINBURNE UNIVERSITY OF TECHNOLOGY

HTTP sessions

Stateful information needs to be maintained a server when it deals with a client.

1. Load page by making an HTTP Request (POST or GET)
2. Server sets cookie and puts sessionID into it. Browser sends it back with each request.
3. Upload info by POST or GET
4. Server compares session ID with it’s list of sessions and “remembers” data for that client.

All vulnerable to sniffing.

HTTP session hijack 2

- ❑ Attacker uses *XSS* to read cookies of authenticated visitors to a site.
  - Attacker can take over a session by writing the received session ID into the attacker’s cookie.
- ❑ Defense:
  - Server side – filter/sanitise input/output
  - Client side – turn off javascript, turn on Application Boundary Enforcer (ABE) privacy plugins (noscript)
  - Https – no protection

XSS

- ❑ Cross-site scripting
  - The use of XSS allows spammers to inject executable code on to forums and bulletin boards, which when executed, download and installs malware, steal cookies (sessions), steal hidden data.

XSS

❑ Reflected XSS attack

- Allows executable html script (javascript, VB script) to be injected by the user into a web application.
- When the application replies with a constructed page, the page includes the executable script.
- Useful for tricking users into allowing script-heavy sites to change browser settings.

❑ Stored XSS attack

- Involves executable script being stored on a server (chat room, forum), which executes when it is displayed by another user.

XSS: DOM-based attack

- ❑ Scripts running in your web browser have access to the browser's DOM (document object model), a hierarchy of objects containing everything displayed and stored on each web page in each instance of the browser.
- ❑ Clever scripting can be used to
  - change the contents of the page, adding options, setting default selections.
  - echo/send private data to 3rd parties (similar to stored and reflected attacks).
  - Access the contents of other browser windows/tabs - largely impossible since Google introduced tab sandboxing.

XSS: DOM-based attack

- ❑ DOM-based attacks avoid storage or transmission of javascript. The injected script is not sent to the server, so it is not detected by automated XSS detection.
- ❑ Writable HTML 2 DOM objects:
  - document.location
  - window.location
  - document.url
  - document.urlencoded
  - document.referrer

jQuery injection

- ❑ jQuery simplifies client-side web dev.
- ❑ Code is very compact, very powerful
  - Easy to hide malicious includes and function calls.
  - Can reveal hidden page contents (passwords, hidden fields, secrets)
- ❑ Solutions:
  - code inspection, sanitise
  - avoid eval(), html(), other unsafe functions

XSS

- ❑ Classic example:
  - Any html which echoes user-provided text back to the browser without sanitizing it is vulnerable
  - http://www.myforum.org?name=<script>alert(document.cookie);</script>
- ❑ Solutions of server side
  - Sanitize / validate all input and output.
- ❑ Solutions of client side
  - disable javascript
  - noScript plugin for Firefox
  - other plugins for other browsers
  - Use a serious sanitizing library

  https://github.com/angular/bower-angular-sanitize

Sandbox

- ❑ A virtual container which restricts the rights of a program.
  - e.g. Program X not allowed to write to disk
  - Sandbox contains a virtual disk (which program X can write to)
  - Sandbox and virtual disk are deleted when Program X terminates.
  - Built into some browsers (Chrome), plugins (Adobe), javascript, Java applets.
  - 3rd party sandboxes (Sandboxie)
  - Virtual machines are the ultimate sandbox
- ❑ But weakened by usability features (unity, vmtools)

Sandbox

❑Sandbox escape exploits exist for many sandboxes.

❑Sandbox escape:

1. Sandboxed (untrusted) application runs code in sandbox (e.g. browser runs js)
2. Untrusted (sandboxed) script runs trusted application from OS
3. Trusted application runs untrusted application from OS

Sandbox example

- ❑ Excel + Flash + download. 10/09/2017

https://threatpost.com/patched-flash-player-sandbox-escape-leaked-windows-credentials/127378/

1. Normally Flash applications on web sites run in the default ‘remote sandbox’ – can’t run code outside sandbox.
2. User clicks on a link which downloads an Excel file.
3. Excel file contains and launches a Flash app.
4. Flash app runs with ‘local with networking’ privileges.
5. Flash app downloads malware and launches it outside sandbox.

Sandbox example

❑ iOS sandbox escape and Priv-esc 1/08/2017

https://www.exploit-db.com/exploits/42407/

1. Set flags in a user-defined message (serialized object) to use shared memory
2. Return value contains a pointer to the shared memory
3. Call 2 function on the serialized object in rapid succession. The first allocates memory, the second sets the data type which will use the memory.
4. A race condition allows the type to be changed after the memory is allocated, allowing a buffer overflow to occur.
5. Send over-sized string to memory, including shell code (the exploit) – runs as root!

XSS

❑ More Solutions:

- Server side: Use **Content Security Policy**
- Set in web server config file
- Apache:

Header set Content-Security-Policy "*<detailed permissions here>*"

COS80013 Internet Security

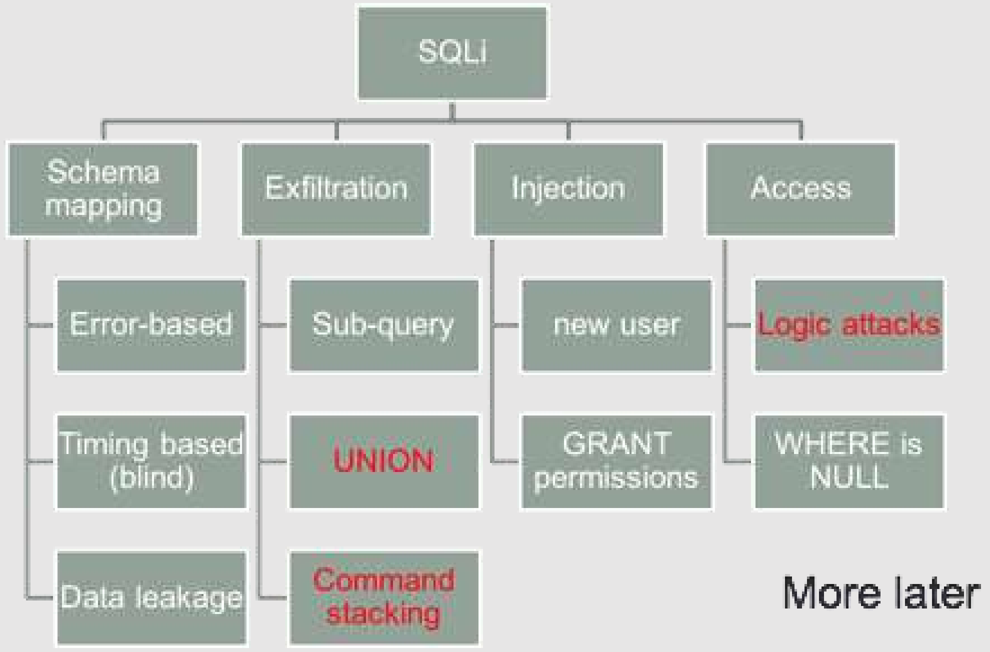
Lecture Week 8B

SWINBURNE UNIVERSITY OF TECHNOLOGY

SQL injection

- Input from web form pasted directly into an SQL command string
- String sent to database server and executed.

Types:



Passthrough functions

- ❑ php and MySQL have functions which let you pass values (e.g. filenames) through to the operating system.
- ❑ If not carefully filtered, these allow a user to read, write, upload and download files.
- ❑ Defense: Restrict rights of webserver files to read or execute only. Web server must not run as root.
- ❑ Validate/filter input.

XML / SOAP

- ❑ SOAP is an XML protocol for transmitting data and executing functions on remote objects through port 80.
- ❑ It allows application developers allow *remote code execution* despite the port blocking that sys. admins. use to prevent hacking.
- ❑ Shell scripts (OS commands) can be wrapped in XML, passed though port 80 as web traffic, and executed on the server.

Best Practices: Web Security

- ❑ Filter
- ❑ Sanitize
- ❑ Validate
- ❑ Both client side **and** server side (Defence in depth)
- ❑ Client side – use well-proven scripts
- ❑ Server side – look-up tables (for validation), filter/sanitise for SQL, BASH, Perl, DOS scripts
- ❑ Restrict Database privileges (later lecture)

Best Practices: Web Security

- ❑ Scripting pages and connection strings must be stored in a web-inaccessible directory. Only the scripts should be able to access the source code of other scripts. Web users must only be able to execute scripts and read html pages.

- ❑ Safe quotes" and other forms of executable script filtering must be enabled.
  The following characters must be filtered or escaped:
  " ' ; - > < ? @ & = (more)

Best Practices: Web Security

- ❑ Only one connection string must be used. It should be stored in a single file and be included into php/asp/jsp/cgi pages as needed. It must not be readable by web users or be stored in a script file. Ideally it should be in a directory which is not accessible to web users.
- ❑ Only the web server user shall be able to access the database. The web server account must be limited. The root password for the database must be set to a non-default value.

Best Practices: Web Security

- Uploadable content (from forums, chat or discussion boards) must be stored on a server with a different domain name to the one used to serve pages (see youtube for an example). This prevents uploaded user's javascript from accessing the javascript functions from the main web domain.
  - –-Enforces javascript *same-origin policy*.

Best Practices: Web Security

- ❑ Transaction systems must not be hosted on a shared host - the same box as other services or other company's web sites/transaction systems/databases.
- ❑ Directory browsing must be disabled.
- ❑ Text input by the user must NEVER be echoed back to the user without filtering (XSS). This includes URL requests, file names, user names, passwords, and all other forms of text or numbers.

Best Practices: Web Security

- ❑ Passwords should not be sent in plain text – use a client-side hashing script or pre-shared encryption key to obfuscate passwords.
- ❑ Or use TLS/HTTPS
- ❑ Passwords should not be stored in plain text.
  - Store password hashes.
- ❑ SQL strings should not be sent.
  - Upload parameters and call stored procedures (pl/SQL) and pass the parameters to them.

Best Practices: Web Security

- ❑ Mixed content including IFrames, mashups, InnerHTML and Frames should not be used.
- ❑ Debug mode and echoing diagnostic messages to the public internet must be disabled on production servers. Such techniques must only be used when the servers in question are not connected to the public internet.
- ❑ No information about the server technology should be revealed to the user. No "row inserted" messages, references to "production server" and so on.
  - – No stack traces!

Best Practices: Web Security

- ❑ Don't use a server as a normal desktop - i.e. don't run client-side programs, check e-mail or surf the web from a server. Don't install consumer-grade software such as MS Office.
- ❑ Data passed to script pages (php, asp) should be sent using the POST method. GET should only be used for navigation. On the server side, only information from the Request.Form() collection should be used.

Best Practices: Web Security

- ❑ Login state should be maintained by session variables. These use cookies but are volatile.
  - $_SESSION['user'] = $user;
  - $_SESSION['loggedin'] = '1';
  - Set secure bit on cookie (https only)
- ❑ The use of cookies to record user data should be discouraged. Unexpected input (e.g. strange URL parameters) should cause the session to be dropped; *i.e.* logout and redirect to the home page.
- ❑ Certificates (for https) should be up to date. Expired certificates train the user to ignore security warnings.