DV2577 Web Security Cheat sheet

Reconnaissance

Includes two phases, **scanning** and **enumerating**.

Injection

Programming languages and libraries without a clear distinction between code and data are more likely to be vulnerable to **injection** attacks.

One mitigation against SQL injection attacks is **input validation using an allow list**.

SQL queries based on user input are more vulnerable to injection attacks than for example session ids and registry keys.

An attacker can directly control the contents of GET and POST parameters as well as `window.location` (JavaScript). It is much harder to control server configuration files, ports and network resources.

An *escaping technique* can be used when you want to tell an interpreter that input is data and not code, or when user input is echoed back to the user in HTML.

The best ways of protecting against injection attacks are **escaping techniques**, **block lists** and **allow lists**.

One of the most common consequences of an injection attack is **denial of service**, another is **data loss**.

A common string to identify a SQL injection vulnerability is `"'!\;#--`.

If the results of an injection attack is not visible to the attacker, it is called a **blind attack**.

SQL injection example

Given the following three queries submitted through a login form, what is the result if ran successfully?

1.

```
SELECT * FROM Users WHERE username='admin' AND password="AND email LIKE
'%@testers.com'
```

2.

```
SELECT * FROM Users WHERE username='admin' -- AND password=" AND email LIKE
'%@testers.com'
```

This query will comment out the `AND` statement and therefore select any user with the username `admin`. This injection will be successful.

3.

```
SELECT * FROM Users WHERE username='admin' ?AND password='' AND email LIKE
'%@testers.com'
```

Cross-Site Scripting (XSS)

One way of mitigating a DOM-based XSS attack is to validate any input that comes from another web site.

To ensure that JavaScript code in a browser cannot access a cookie, it must be marked with the **HttpOnly flag**.

To ensure that only sites served over HTTPS can access a cookie, it must be marked with the **secure flag**. Note that the cookie itself can still be created over HTTP.

A common string to identify a XSS vulnerability is `<script>alert();</script>`.

Same-origin policy

Under this policy, a web browser will allow scripts within the web page to access data in a second web page if and only if both web pages have the same origin. The origin is defined as the combination of protocol, hostname and port number.

The idea of a same-origin policy is to prevent malicious scripts from accessing sensitive data on another web page.

Consider a user on `http://www.example.com/dir/page.html`.

Requests to and from the following URLs will be successful:

- `http://www.example.com/dir/other.html`
- `http://www.example.com/dir/page2.html`
- `http://www.example.com/dir2/other.html`
- `http://username:password@www.example.com/dir2/other.html`

Requests to and from the following URLs will not be successful:

- `http://www.example.com:81/dir/other.html`
- `http://en.example.com/dir/other.html`
- `http://v2.www.example.com/dir/other.html`

XSS Payload example

Given the following HTML form:

```
<form name="login">
  <label for="user">Username:</label>
  <input type="text" id="user" name="user" />
  <label for="password">Password:</label>
  <input type="text" id="password" name="password" />
</form>
```

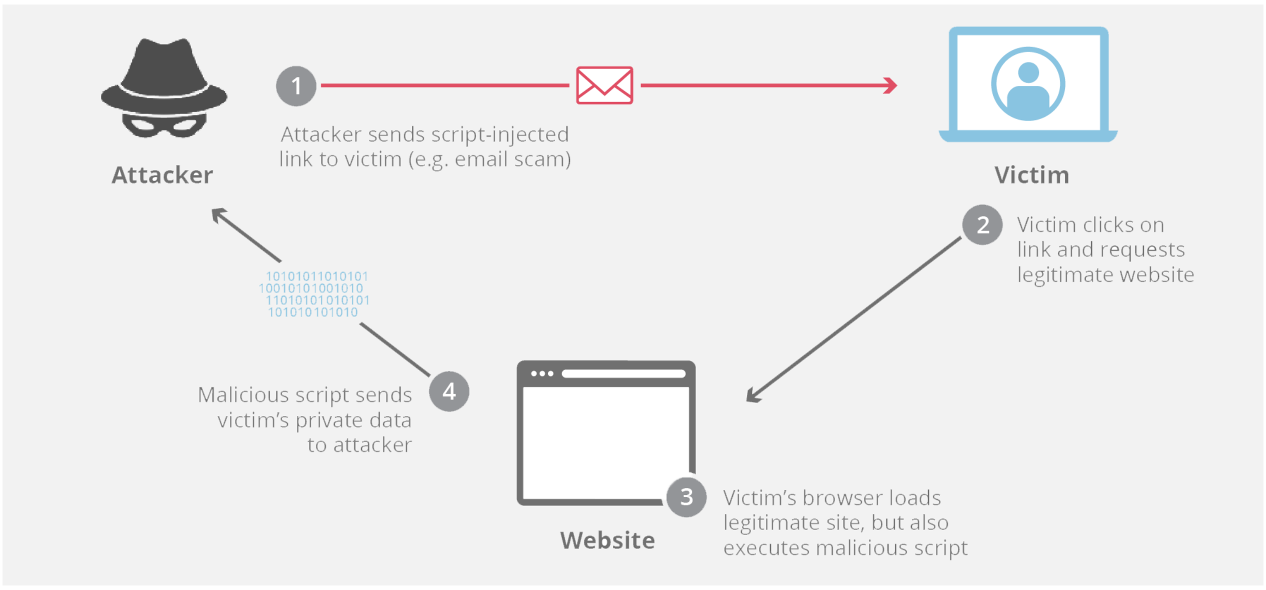
The following scripts will access the credentials and show it on screen.

```
alert(document.getElementById('user').value,
document.getElementById('password').value)
```

```
alert(document.querySelector('#user').value,
document.querySelector('#password').value)
```

```
alert(Array.from(document.querySelectorAll('input')).map(x =>
`${x.name}:${x.value}`).join('\n'))
```

Reflected XSS diagram


Attacker
1
Attacker sends script-injected link to victim (e.g. email scam)
Victim
2
Victim clicks on link and requests legitimate website
Website
3
Victim's browser loads legitimate site, but also executes malicious script
4
Malicious script sends victim's private data to attacker
10101011010101
10010101001010
11010101010101
101010101010

Authentication / Authorization / Access control

Roll Based Access Control (RBAC) is a way to restrict file access based on the user or group identity and security classification of the information.

Attribution Based Access Control (ABAC) is considered a next-generation policy-based access control system.

Cross-Site Request Forgery (CSRF) attacks can be mitigated by implementing a defense-in-depth technique such as not including secrets in the URL.

CAPTCHA can be used to protect authentication systems from automated brute-force attacks.

If poorly implemented custom code or misconfigured off-the-shelf code is used, one is more likely to end up with broken authentication or session management vulnerabilities.

In an authentication system **strong passwords**, **user logout and session inactivity** and **encrypted and hashed credentials** are **mandatory**.

The most common consequences of a CSRF attack is **elevation of privileges**, **denial of service** and **spoofing**.

A common string to identify a broken authorization vulnerability is `http://example.com/login?userid=1337` where you try different values for the user id.

If you are intercepting a web application, log in and find that it makes a request to `http://www.example.com/app?action=login&uname=jane&password=123456`, then the site incorrectly uses GET to send credentials. The site is also susceptible to CSRF attacks as the complete form is available as a URL. The page is served over HTTP which is never a good idea when sending credentials.

Configuration / Code

If you have not yet applied the most recent service packs and updates to a web application you are more susceptible to security misconfiguration threats.

Transport layer protection is best implemented by enabling TLS (previously SSL).

In order to combat a Denial of Service (DoS) attack, one can install `mod_evasive` for Apache and configuring a lockout time period whereby an IP is banned for some time if too many login attempts are made.

When configuring a secure web server, one should review all settings and configuration as well as disabling unnececssary features.

A web server will display a directory listing if there is no `index` file such as `index.html` or `index.php` in a directory, depending on what's configured.

A common string to identify a path traversal vulnerability is `../../../../../../../../interesting/path`.

A common string to identify a local file inclusion vulnerability is pretty much the same as the above, with the interesting path being that of a file.

A common string to identify a remote file inclusion vulnerability is `www.example.com/?page=http://example.com`.

If a URL `http://example.com/home/index.aspx?country=Ua` works, but `http://example.com/home/index.aspx?country=foo` gives the error `*Could not open file: D:\app\default\home\logs\foo.log (invalid file)*`, the application is likely vulnerable to a **local file inclusion**. This could be fixed by having a lookup table for allowed values for `country`.

One can stop web spiders from crawling certain directories by using the `robots.txt` file which is respected by well-behaved spiders. Some spiders do not respect the file, but does send a user-agent with each request which can then be used to identify and block the spider.

mod_security

```
# Start monitoring requests
SecFilterEngine On

# Enable scanning of the request body
SecFilterScanPOST On

# Prevent path traversal (..) attacks
SecFilter "\.\./"

# Weaker XSS protection but allows common HTML tags
SecFilter "<( |\n)*script"

# Prevent XSS atacks (HTML/Javascript injection)
SecFilter "<(.|\n)+>"

# Command execution
SecFilter /etc/password
SecFilter /bin/ls

# Very crude filters to prevent SQL injection attacks
SecFilter "delete[[:space:]]+from"
SecFilter "insert[[:space:]]+into"
SecFilter "select.+from"

# Information leakage
# Start filtering request output
SecFilterScanOutput On
# Filter out common errors
SecFilterSelective OUTPUT "An Error Has Occurred" status:500
SecFilterSelective OUTPUT "Fatal error:"
```

Payment Card Industry (PCI) framework

The PCI framework provides **guidance for securing payment card data**. PCI provides an actionable framework for developing a robust payment card data security process, including **prevention**, **detection**, and **appropriate reaction to security incidents**.

Session manipulation / hijacking attack

Techniques for session hijacking:

- Man-in-the-middle attack
  - A man-in-the-middle can intercept traffic and see / change the session of a victim
- Client-side attacks (XSS etc.)
  - A XSS vulnerability could allow the attacker to send off cookies to a server run by the

attacker
- Predictable session tokens can allow for forged tokens