HACKING CHEAT SHEET

Reconnaissance

Map Out The Attack Surface Parts Of The Range May Be Hidden. These Techniques Will Help You Find Them.

Check Robots.txt

The robots.txt file, found in a site's web root, tells well-behaved web crawlers what parts of the site to ignore. You're not a well-behaved web crawler, so you can look at those pages. You may find pages the rest of the site doesn't link to.

Try Some Common URLs

By guessing common page and directory names, you might be able to discover even more content. A tool like dirbuster can help (but it's probably overkill here).

Look For HTML Comments & Hidden Elements

Look for forms, form fields and links that appear in the page source, but aren't visible on the page. The CSS style `display: none;` hides an element; remove the styling to make it visible again. Take a look at the HTML comments too!

System Fingerprinting Identify What Components The System Is Using.

Questions To Ask

- Which web server - Apache, nginx, IIS?
- Which web framework - .NET, Django, Struts?
- Which database - MSSQL, MySQL...?
- Version numbers for web server and other components - are they up to date?
- How do they handle session management? Did they use a framework or roll their own?

Where To Look

- **HTTP response headers** - look for `Server` and `X-Powered-By`
- **Error messages** - look for version Info and stack traces.
- **Cookies** - cookie names can reveal framework info. If they're managing cookies themselves, think about how they're being generated. Are they predictable? How are they processed on the server?

Open Source Intelligence (OSINT) Gather Information On The Public Internet

What To Look For

- Known vulnerabilities in frameworks/other components
- Default credentials
- Employee contact info/personal information

How To Find It

- Google error messages, cookie names, version headers, password hashes...
- Read framework/component documentation
- Read framework/component security advisories
- Look up company employees on social media

> Your Google searches aren't private! When testing real applications, don't Google password hashes or other highly sensitive information - not even in incognito mode.

Attacks

Cross-Site Scripting (XSS) Inject Malicious JavaScript Into A Webpage

XSS allows an attacker to inject client side code (HTML, JavaScript, etc.) into the page such that it is rendered in the victim's browser. XSS is possible when user input that hasn't been properly output-encoded is included in a web page. XSS can come in three types:

- **Reflected:** script is provided by the caller, included in the response from the server, and executed in the browser. E.g. a search term provided in a URL parameter is rendered in the body of the page.
- **Persistent:** script is stored in a datastore and included the body of the page when it is rendered. E.g. a forum that allows users to leave comments for one another.
- **DOM-based:** script is included in the page via client-side JavaScript rendering. The malicious input is never sent to the server. E.g. an application that updates a user's display name client-side based on provided input. (example.com/welcome/#name=Bob)

What To Look For

Data or text you provided is reflected back to you on the page. If that data isn't properly encoded, an XSS vulnerability might be present. Examples:

- You search for "foo" in the range's search form. The results page says, "Your search for 'foo' returned no results""
- You navigate to example.com/fakePage. The 404 page says, "Sorry, page "fakePage" not found."
- You try to log in as user "bob" and the following error message is returned: "The user 'bob' does not exist."

Test Cases - Discovery

Once you find a place where your input is being reflected back to you, try one of these tests cases to see if it's being properly encoded.

- `<marquee>` - if not properly encoded, causes content to scroll sideways across the page.
- `<plaintext>` - if not properly encoded, causes browser to display raw HTML

Test Cases - Exploitation

To score points for XSS, your exploit must open up a JavaScript alert box.

- `<script>alert(1)</script>` - the simplest possible XSS exploit.
- `<img src="javascript:alert('xss')"/>` - some applications filter `<script>` tags. This is one way to bypass that.
- `<body onload="alert('xss')"/>"` - event handlers like `onload` are another XSS injection point.

SQL Injection Execute Arbitrary SQL Commands On The Server. Possible When The Server Concatenates User-supplied Data With SQL Code.

What To Look For

Look for places where the application could be querying a SQL database. If a page isn't completely static, it's probably retrieving information from a database. Think about what SQL code the application might be running. For example, when a user logs in, the SQL query might look like:

```
SELECT \* FROM Users
WHERE Username= '\[user input\]'
AND password = '\[user input\]';
```

Special Characters

- `'` - a single quote delimits strings in SQL queries. Because most user input is wrapped in strings, this is usually your first step to breaking out of the string and changing the rest of the query.
- `#`, `--` - comment signs. A comment sign tells the SQL interpreter to ignore the rest of the line.
- `;` - a semicolon ends a SQL comment. This can be used to string multiple SQL commands together if the database supports it.
- `OR`, `AND` - SQL supports boolean operators.
- `<`, `=` - SQL supports comparison operators. Note that comparison uses `=`, not `==`.

Test Cases

- `'` - a single quote is the simplest discovery test case. If that throws a SQL error, it's a sure sign you've found SQL injection.
- `\'` - another discovery test case. Some databases escape special characters with backslashes. If the application escapes unsafe characters, this may bypass it.
- `' OR 1=1#` - inserted into a `WHERE` clause, this forces it to evaluate to true. Good for bypassing authentication on login forms.

Forceful Browsing

Pages may just be hidden from users, not protected with proper access control checks. Forceful browsing may just mean browsing directly to an administrative page, or you might need to combine it with parameter tampering to view pages that you shouldn't have access to.

What To Look For

Keep an eye on the URL. As an administrator or other privileged user, try browsing to as many authenticated pages as possible. Copy each URL to be used later.

Test Cases

Try to access each authenticated page (including any URL parameters) as an unauthenticated or lower-privileged user. If a page contains information that only one user should be able to access, log out, log in as another user, and try to browse to the page again.

Parameter Tampering

It's possible to change any form or URL parameter. Sometimes changing a value (like a user ID) can allow you to see or tamper with data without authorization.

What To Look For

All parameters sent to the server:

- URL parameters
- Dropdowns
- Checkboxes
- Radio Buttons
- Hidden form fields
- Cookies

Test Cases

To tamper with a URL parameter, edit it in your address bar, then hit enter. To tamper with a form parameter, edit it using your browser's developer tools, then submit the form.

File Upload

There are lots of ways to abuse insecure file uploads. An attacker can upload code, then look for ways to execute it on the server. They can overwrite other files on the server. Or they could upload malicious scripts or malware that will execute when other users view them.

What To Look For

Any file upload functionality.

Test Cases

Tamper with file names, paths, and extensions. Can you upload a file type that isn't permitted? Can you upload code and run it?

More Tools & Resources

- SQL Injection Cheat Sheet
- OWASP XSS Filter Evasion Cheat Sheet

- ASCII to Hex - General-purpose text encoding/decoding
- CrackStation - online password hash cracker
- hashcat - offline password hash cracker