cmd+ctrl
A SECURITY INNOVATION CYBER RANGE

getting started

This document has been designed to be a reference to help you get started in your white-hat hacking journey. There are a few basic techniques and attacks that are important to know about when getting started with security testing.

start hacking!

Once you're logged in, click the **Launch Panel** button at the top which launches InstaFriends Social Media Web site. Each player has their own instance so no need to worry about your play effecting others - or vice versa. A Virtual Machine is set up just for you, but that takes about 2 minutes, so be patient.

techniques

Security Testing is a conversation. There is no one way to conduct a conversation, and likewise there is no one way to conduct a security assessment. Think of things you'd like to discover about the application, then think of questions that you can ask to find out the answers. A good security tester is both **observant** and has a **good memory.** Be sure to see everything you can and remember as much as possible. You'll never know when something you saw hours or days ago will come up later. You must also have a **good imagination.** Think about how the application was developed, what assumptions were made, and how those assumptions can be exploited.

observation

The application discloses an enormous amount of information in everyday operation. Create a complete picture of the application, then use that knowledge to attack it. Here are a few things that you'll want to discover about the site you're testing:

- **Web Server type and version number** - Google for known vulnerabilities, default settings, and more.
- **Error Messages** - Where did the error message come from (Database, web server, 404 page, javascript, input validation routine)? What can you learn from the error message (software, configuration, attack vectors)?
- **Source Code** - Can you find the source code of the application somewhere? This can give you a significant inside track. Can you use other vulnerabilities to download the source? Can you discover a jar or war file and decompile it with a tool such as JAD?
- **Client Side Code** - HTML, HTML Comments, JavaScript, and CSS can all give you critical information about the application. Right click and view source!
- **Input Validation** - Input validation may be possible to bypass if it is done client side only.
- **URL** - The URL is a veritable font of information . Look at the parameters , domain , directory , and other clues to discover how the application does what it is doing.
- **Cookies** - What do the cookies look like? Are all of those cookies standard? Are they set *secure* or *httponly*?
- **UI** - Observe any time a UI element is out of place, this could be an indication of new (or old) or buggy added functionality.

the conversation

Each time you learn something new about the application think about what doors that new answer opened for you. If you discover what OS it's using, you can now google for known vulnerabilities, configurations, directory structures, or other information that you can use.If you start learning about the application, what does that tell you about the assumptions that the developer made?

- Does validation always occur on the client?
- Is the validation different on the server?
- Was an ORM used? Was it used consistently?
- Are there comments in HTML? What do they say?

Once you discover one thing about the application, think about what the next question can be to learn more.

vulnerability chaining

Frequently it isn't possible to get what you want in a single vulnerability, so we have to chain multiple vulnerabilities together.
A Cross Site Scripting (XSS) vulnerability may allow you to bypass the Cross Site Request Forgery (CSRF) protection, which allows you to trick a user into clicking a link to transfer money.
Once you've found a few vulnerabilities think about how you can chain multiple issues together to accomplish something greater!

attacks

cross site scripting (xss)

XSS allows an attacker to inject client side code (HTML, JavaScript, etc.) into the page such that it is rendered on the client.

- Reflected - script is provided by the caller and executed in the browser, e.g. an error message that is rendered in the body of the page which has been provided as a url parameter that is not properly encoded on the server.
- Stored - script is stored in a datastore and added to the body of the page as it is rendered, e.g. a forum that allows users to leave comments for one another.

things to look out for

Any time data or text that you have provided is reflected back to you, there is a possibility for XSS. When you discover a potential injection point look at the context. Different attacks may be possible if your attack string is in HTML, JavaScript, a header, a link or somewhere else.

example

Consider you attempt a login with username “testAccount” and you see an error message
“Sorry there is no account called testAccount in the system.”
This tells us a few things about the system.

1. It is performing user and password checks separately
2. It is leaking information about valid users (Information Disclosure Vulnerability)
3. It is reflecting the username provided back to you which may be an injection point. Attempt the test cases below.

test cases – discovery

- `<marquee>` or `<plaintext>` - Frequently an opening marquee or plaintext tag can be very useful in flushing out potential areas for XSS injection. The attack string is very obvious when not properly validated or encoded.
- `'';!--"<XSS>=&{()}` - This XSS locator string includes a large number of potentially dangerous characters that should always be encoded. Insert this string into the page and search for “XSS” in the page, validate that each of the characters have been properly validated for context.

test cases – exploitation

- `<script>alert(1)</script>` - this is the most basic attack string. It will simply pop up an alert box if executed. Frequently this will not work because of basic blacklist filtering on the `<script>` tag.
- `<IMG SRC="javascript:alert('XSS');"> - This uses the image tag to execute javascript which can bypass simple script tag checks.`
- `<BODY ONLOAD=alert('XSS')>` - Many other tags can be used along with javascript events to execute an attack string.
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet - There are far more ways to execute JavaScript on a website than we have space for here. Go to the URL above for more examples.

information disclosure

Information disclosure can help you learn about the application quickly. It can also be a security vulnerability in and of itself. Stack traces, information about the server, data quality responses, and any other information that could be useful to the attacker should be removed from production.

things to look out for

- **Server Headers** - Search google for any server information you can find.
- **Error messages** - Think about what this error message tells you about the application and where the error message came from.
- **Stack Traces** - These allow you to learn more about the application and how it works.

test cases

Information Disclosure issues are usually discovered during the course of other testing. Every other test case in this document may provide you information.

sql injection (sqli)

SQL injection vulnerabilities allow an attacker to execute arbitrary SQL commands on the server. This is due to the server failing to properly separate user supplied data from SQL code.

Frequently simply typing a `'` (single quote) into text boxes will cause SQL error messages (Information Disclosure) to occur which can give you more information about the system and how to attack it. Look out for these types of error messages to know you’re on the right track. Also think about what the SQL code may look like on the server.
Logging in may be performed by executing the following SQL code:

```
SELECT * FROM Users
WHERE Username = '[userprovided]'
AND Password = '[userprovided]';
```

Typing in a single quote into the username field will cause this SQL command to be invalid, which will cause an error message. You may be able to bypass the authentication of this system by adding the right SQL command.

test cases – special characters and commands

- `'` - a simple single quote is the first test case to use to discover SQL injection
- # - the # is a SQL comment and tells the sql interpreter to stop executing the rest of the line
- `;` - the semicolon is the end of a command. This may be used to string multiple SQL commands together if supported by the database.
- `OR`, `AND` - SQL supports logical operators such as or and and
- <, = - SQL supports comparison operators

test cases – exploitation

- `# OR 1=1 --` - this statement closes a string and 1=1 always resolves to true. Any statement or’d with true is always true. This means the rest of the statement will be true. Good for bypassing authentication and logins.

parameter/url tampering

Parameters when passed in the URL bar or provided by the client may be tampered with. Often times these parameters are assumed to be immutable and are not validated on the server. Things to look out for: Hidden values, Dropdown values, Checkboxes, Radio Buttons URL parameters

test cases

Look at the source code to discover parameters, then use the Firebug plugin to manipulate the parameters before submitting any forms.

file upload

Uploading arbitrary files can be problematic depending on the context in which they can be accessed. Uploading an image may be relatively benign, but if the attacker can upload script or executable files they may be able to execute arbitrary code on the server or use the server as a malware repository.

things to look out for

Anywhere a file can be uploaded could be vulnerable.

test cases

Attempt tampering with file names, encoding, path, extension, or contents may cause the server to interpret the file differently. Attempt uploading a JSP file to the server then browse to that file. Does the upload succeed and does the file execute?