Web Exploitation 101

Breaking OverTheWire: Natas

Jayrup Nakawala

The Rules of the Game

Welcome to Natas

Natas teaches the basics of server-side web security.

`https://overthewire.org`

The Goal: Find the password for the next level.

The Reality: You are exploiting lazy developers, bad configurations, and absolute trust in user input.

The Attacker’s Mindset

Forget the GUI. The browser is lying to you.

If you want to break a web app, you need to look at what the server is *actually* sending, not what the browser is rendering.

We are going to use exactly what we learned in Term 1: **DevTools and cURL.**

Level 0 & 1: The Frontend is a Lie

Natas 0: Hide and Seek

The Setup: A plain web page. You need the password.

The Mindset: Developers leave comments in their code. The browser hides them. You shouldn't.

The Execution: Right-click -> View Page Source (or `Ctrl+U`). Read the raw HTML. The password is right there in a comment.

Natas 1: The Illusion of Control

The Setup: Same as Level 0, but “Right-Click has been disabled!”

The Mindset: Client-side JavaScript restrictions are a joke. You control your machine, not the server.

The Execution: Bypass the UI entirely. Use a terminal:

```
curl -u natas1:PASSWORD
http://natas1.natas.labs.overthewire.org
```

The server blindly hands you the source code.

Level 2 & 3: Information Leakage

Natas 2: Open Doors

The Setup: There is a hidden pixel image on the page.

The Mindset: Where do images live? In directories. If a server isn't configured to block directory listing, you can browse it like your own file system.

The Execution: Check the image URL (`/files/pixel.png`). Navigate to `http://.../files/`. Find the `users.txt` file sitting wide open.

Natas 3: The Secret Map

The Setup: “There is nothing on this page.”

The Mindset: How do developers tell Google *not* to index their secret folders? They write a map of all their secrets and put it in the root directory.

The Execution: Read the `/robots.txt` file. It literally says: `Disallow: /s3cr3t/`. Go to `/s3cr3t/` and take the password.

Level 4 & 5: Manipulating State

Natas 4: Forging Origins

The Setup: “Access disallowed. You are visiting from”” while authorized users should come only from “http://natas5.natas.labs.overthewire.org/”.”

The Mindset: How does a server know where you came from? The `Referer` HTTP header. Remember Term 1: HTTP is just text. You can forge it.

The Execution: Use a browser extension or cURL to inject the header: `curl -u natas4:PASS --referer "http://natas5.natas.labs.overthewire.org/" http://natas4.natas...`

Natas 5: Trusting the Client

The Setup: “Access disallowed. You are not logged in.”

The Mindset: HTTP is stateless. To remember you, the server gives you a Cookie. If the cookie says `loggedin=0`, what happens if you just... change it?

The Execution: Open DevTools -> Application -> Cookies. Change the value of `loggedin` from `0` to `1`. Refresh the page. You are the admin now.

Level 6 & 7: Backend Flaws

Natas 6: Logic vs. Data

The Setup: A form asking for a secret. The PHP source code is provided.

The Mindset: Read the code. The PHP logic checks your input against a variable stored in an `include` file.

The Execution: The code says `include "includes/secret.inc"`. Navigate directly to `/includes/secret.inc`. The server hands you the raw plaintext secret. Submit it to the form.

Natas 7: Escaping the Sandbox

The Setup: A webpage with a URL like `index.php?page=home`.

The Mindset: That `page` parameter is taking user input and asking the server to load a file. What if we ask it to load a system file instead of a web page? This is Local File Inclusion (LFI).

The Execution: Break out of the web directory. Change the URL to `index.php?page=/etc/natas_webpass/natas8`. The server blindly reads the password file and serves it to you.

Your Turn

Homework: Natas 8-10

You know the mindset. Now apply it.

- **Natas 8:** Reverse engineering basic PHP functions (hex, base64).
- **Natas 9:** Command Injection. The server is running a bash command. Hijack it using `;`.
- **Natas 10:** Bypassing weak input filters.

Don’t just look up the answers. Read the source code. Understand *why* it breaks.

Questions?