George, Levi

CS44500 – Computer Security

02/24/23

Lab 5 – XSS Attacks

ENV SETUP.

I open up etc/hosts file using root permissions to write to it.

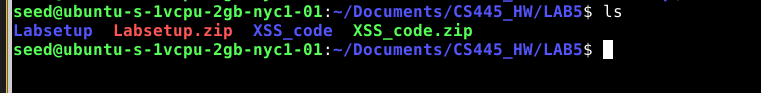


I add the entries

Text

Description automatically generated

I curled the labsetup file and unzipped it.



These files were inside



These are the contents of the .yaml file included for composing in docker.

Text

Description automatically generated

I ran dcbuild to compose my containers.

Text

Description automatically generated

I ran dcup to start the containers

Text

Description automatically generated

I successfully navigated to Elgg.

Graphical user interface, text, application, email, website

Description automatically generated

Lab Tasks

TASK 1 – Posting Malicious Message for Alert Windows

I entered the generic script <script>alert(“XSS”);</script>

Into Boby’s brief description, pressed submit on the edit profile page and got this.

Even simply going to the member’s page loaded the malicious script since it was in the brief description field.

Graphical user interface, application

Description automatically generated

TASK 2 – Posting a Malicious Message to Display Cookies

I replace my previous script with a new one for cookies.

A picture containing application

Description automatically generated

I save and get this for the cookies.

Graphical user interface, text, application

Description automatically generated

TASK 3 – Stealing Cookies from the Victim’s Machine

I set netcat to listen according to the lab instructions

Text

Description automatically generated

I insert the img, with the src on another server (that being the malicious host). I reload the page after saving the malicious script.

Text

Description automatically generated

TASK 4 – Becoming a victim’s friend



During my initial investigation, I found the structure follows this

We get the ID, a “TS”, and a token.

So, we need the TS, Token, and Samy’s ID for adding the friend

To get Samy’s ID, I added Charlie and Samy as friend’s on Boby’s account so I could verify that the friend=# is actually the number of the user we are adding. Something similar could be accomplished via making a second account and adding Samy to find that ID (if we were actually initiating an attack without all of the account information)

Samy’s ID is 59 and Charlie’s is 58

Graphical user interface, text, application, email

Description automatically generated

I will use the above get request to form my malicious payload

Text, timeline

Description automatically generated

I appended Samy’s friend id to the URL then the TS and Token in that order.

I saved the malicious About Me script.

I logged in as Alice and visited Samy’s page, at first it didn’t show him as a friend, but when I attempted to remove him, he was re-added after a short visit to his page.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

A picture containing text

Description automatically generated

Question 1 Response: They are tokens, a way of identifying the request as coming from an internal and authorized system within a specified time frame, so someone can’t build and send a malicious post request some time from when the initial conversation occurs (programmatically speaking)

Question 2 Response: No, not this same attack, we could load the script from a different location in something like the brief description, but attempting to use this same vector would fail since it would treat the formatting of the tags differently. i.e. the ;lt; and ;gt; symbols replace each of the symbols around our tags.

TASK 5 – Modifying the victim’s profile

While I was still in Alice’s account, I used the HTTP Header tool to grab an edit POST

Text

Description automatically generated

The first thing I notice is that we have each attribute in the message followed by the GUID at the end.

Below is Alice’s entry before my attack.

Graphical user interface, text, application, email

Description automatically generated

I put the following code into Samy’s About page.

The code below essentially lets us put together a payload.

We have the name of the user, their ID, timestamp and token. Then we can put all of those into our content which we can also put the profile setting that we want to alter. At the end we include the guid since that is how it is structured.

Then we form an asynchronous http request to send to the server with the malicious payload, allowing us to submit profile changes without the authorization of the user.

Text

Description automatically generated

After logging in as Alice and visiting Samy’s page I found that Alice’s description changed, revealing her true feelings toward Samy.

Graphical user interface, text, application

Description automatically generated

The if statement simply stops the request from going off for Samy, if I were to remove that if statement then it would run after I save the changes and then it would reset my own description in this case removing the code. If I could put the code somewhere else then it might work for that particular entry, but if I stored the code in the same entry I am trying to edit on other users then it would fail.

Graphical user interface, text, application

Description automatically generated

Here is the edited code with the if statement removed.

I saved and reloaded the page then checked Samy’s profile. That if statement just protects the user from screwing up their attack.

Graphical user interface, text, application, email

Description automatically generated

TASK 6 – Writing a Self-Propagating XSS Worm

I essentially took all the code we were given and smushed it into a single file.

Text

Description automatically generated

Text

Description automatically generated

The thing about this code is it performs the same functions as the previous two tasks, but in addition the code will copy itself that is the jsCode variable at the top. Encapsulate itself in a set of script tags the HeaderTag and TailTag variables. Then wormCode hosts the encoded malicious code in itself which we can then post via an HTTP request, much like the previous part where we forced victim accounts to say “I am Samy’s biggest fan”

I created the code and inserted it into Samy’s About Me section.

Graphical user interface, text, application, email

Description automatically generated

I entered the code into Samy’s about me, then visited his page using alice, I found the code was in her about me, then her account had also been manipulated as per my code.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Boby’s account is initially empty.

I visited Alice’s account and had the code injected into Boby’s account

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

TASK 7 – Defeating XSS Attacks using CSP

TASK 7.1 – Lab Tasks

7.1.1 – Describe and explain your observations when you visit these websites

On my initial visit to each of the example sites, I find that each one allows different javascript scripts run. 32a allows all of them to run, but 32b and 32c are a bit more selective.

I am told that 32c is a PHP script which includes it’s own CSP and it is coded to allow Nonce 111-111-111 to run., so it makes sense the first script runs on the 32c page.

32a clearly implements no CSP policies, allowing all scripts to run.

32b implements host based CSP policies, so it makes sense that only it’s script and scripts from other sites that it trusts would work.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

7.1.2 – Click all the buttons and describe your observations.

Graphical user interface, text, application

Description automatically generated

I pressed the button on all three pages and was only able to get a response from example32a. The CSP policies implemented on the Host and Application level have blocked the script, so it isn’t an internal script nor did it come from example70. We can only presume that it came from an external location (example60 perhaps) that doesn’t have a valid nonce.

7.1.3 – Change server configuration for example32b so Areas 5 and 6 run. Show modified config

Text

Description automatically generated

The above is my chosen configuration, below shows it passing section 5 and 6

Graphical user interface, text, application, email

Description automatically generated

7.1.4 – Change the PHP code for example32c to show areas 1,2,4,5, and 6 as OK.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

7.1.5 – Explain why CSP prevents XSS attacks.

CSP acts as a whitelist, we assume all external sources are threats and disallow their scripts from running on our systems. If we can’t run scripts from external locations unless we say so, we prevent a lot of issues that could occur with malicious users attempting to insert code.

The nonces we include would require sources to provide the appropriate passcode of sorts before getting to run their script. This means, our scripts, which just show up may not meet nonce requirements or source requirements and hence would fail to run.

However, there are issues with supply chain attacks. If we compromised example70.com and replaced all the scripts with malicious codes then this website would run our code when it references example70.