README2.md

Índice de subtermas:

6.5 Remote File Inclusion (RFI)

La vulnerabilidad Remote File Inclusion (RFI) es una vulnerabilidad de seguridad en la que un atacante puede incluir archivos remotos en una aplicación web vulnerable. Esto puede permitir al atacante ejecutar código malicioso en el servidor web y comprometer el sistema.

En un ataque de RFI, el atacante utiliza una entrada del usuario, como una URL o un campo de formulario, para incluir un archivo remoto en la solicitud. Si la aplicación web no valida adecuadamente estas entradas, procesará la solicitud y devolverá el contenido del archivo remoto al atacante.

Un atacante puede utilizar esta vulnerabilidad para incluir archivos remotos maliciosos que contienen código malicioso, como virus o troyanos, o para ejecutar comandos en el servidor vulnerable. En algunos casos, el atacante puede dirigir la solicitud hacia un recurso PHP alojado en un servidor de su propiedad, lo que le brinda un mayor grado de control en el ataque.

A continuación, se proporciona el enlace al proyecto de Github correspondiente al laboratorio que estaremos desplegando para practicar esta vulnerabilidad:

- DVWP: https://github.com/vavkamil/dvwp

Asimismo, se os comparte el enlace directo para la descarga del plugin ‘Gwolle Guestbook‘ de WordPress:

- Gwolle Guestbook: https://es.wordpress.org/plugins/gwolle-gb/

6.6 Log Poisoning (LFI --> RCE)

El Log Poisoning es una técnica de ataque en la que un atacante manipula los archivos de registro (logs) de una aplicación web para lograr un resultado malintencionado. Esta técnica puede ser utilizada en conjunto con una vulnerabilidad LFI para lograr una ejecución remota de comandos en el servidor.

Como ejemplos para esta clase, trataremos de envenenar los recursos ‘auth.log‘ de SSH y ‘access.log‘ de Apache, comenzando mediante la explotación de una vulnerabilidad LFI primeramente para acceder a estos archivos de registro. Una vez hayamos logrado acceder a estos archivos, veremos cómo manipularlos para incluir código malicioso.

En el caso de los logs de SSH, el atacante puede inyectar código PHP en el campo de usuario durante el proceso de autenticación. Esto permite que el código se registre en el log ‘auth.log‘ de SSH y sea interpretado en el momento en el que el archivo de registro sea leído. De esta manera, el atacante puede lograr una ejecución remota de comandos en el servidor.

En el caso del archivo ‘access.log‘ de Apache, el atacante puede inyectar código PHP en el campo User-Agent de una solicitud HTTP. Este código malicioso se registra en el archivo de registro ‘access.log’ de Apache y se ejecuta cuando el archivo de registro es leído. De esta manera, el atacante también puede lograr una ejecución remota de comandos en el servidor.

Cabe destacar que en algunos sistemas, el archivo ‘auth.log‘ no es utilizado para registrar los eventos de autenticación de SSH. En su lugar, los eventos de autenticación pueden ser registrados en archivos de registro con diferentes nombres, como ‘btmp‘.

Por ejemplo, en sistemas basados en Debian y Ubuntu, los eventos de autenticación de SSH se registran en el archivo ‘auth.log’. Sin embargo, en sistemas basados en Red Hat y CentOS, los eventos de autenticación de SSH se registran en el archivo ‘btmp’. Aunque a veces pueden haber excepciones.

Para prevenir el Log Poisoning, es importante que los desarrolladores limiten el acceso a los archivos de registro y aseguren que estos archivos se almacenen en un lugar seguro. Además, es importante que se valide adecuadamente la entrada del usuario y se filtre cualquier intento de entrada maliciosa antes de registrarla en los archivos de registro.

6.7 Cross-Site Request Forgery (CSRF)

Si a la hora de hacer el ‘docker-compose up -d‘, os salta un error de tipo: “networks.net-10.9.0.0 value Additional properties are not allowed (‘name’ was unexpected)“, lo que tenéis que hacer es en el archivo ‘docker-compose.yml‘, borrar la línea número 41, la que pone “name: net-10.9.0.0“.

Con hacer esto, ya podréis desplegar el laboratorio sin ningún problema.

El Cross-Site Request Forgery (CSRF) es una vulnerabilidad de seguridad en la que un atacante engaña a un usuario legítimo para que realice una acción no deseada en un sitio web sin su conocimiento o consentimiento.

En un ataque CSRF, el atacante engaña a la víctima para que haga clic en un enlace malicioso o visite una página web maliciosa. Esta página maliciosa puede contener una solicitud HTTP que realiza una acción no deseada en el sitio web de la víctima.

Por ejemplo, imagina que un usuario ha iniciado sesión en su cuenta bancaria en línea y luego visita una página web maliciosa. La página maliciosa contiene un formulario que envía una solicitud HTTP al sitio web del banco para transferir fondos de la cuenta bancaria del usuario a la cuenta del atacante. Si el usuario hace clic en el botón de envío sin saber que está realizando una transferencia, el ataque CSRF habrá sido exitoso.

El ataque CSRF puede ser utilizado para realizar una amplia variedad de acciones no deseadas, incluyendo la transferencia de fondos, la modificación de la información de la cuenta, la eliminación de datos y mucho más.

Para prevenir los ataques CSRF, los desarrolladores de aplicaciones web deben implementar medidas de seguridad adecuadas, como la inclusión de tokens CSRF en los formularios y solicitudes HTTP. Estos tokens CSRF permiten que la aplicación web verifique que la solicitud proviene de un usuario legítimo y no de un atacante malintencionado (aunque cuidadín que también se pueden hacer cositas con esto).

Os compartimos a continuación el enlace al comprimido ZIP que utilizamos en esta clase para desplegar el laboratorio donde practicamos esta vulnerabilidad:

- Lab Setup: https://seedsecuritylabs.org/Labs_20.04/Files/Web_CSRF_Elgg/Labsetup.zip

6.8 Server-Side Request Forgery (SSRF)

El Server-Side Request Forgery (SSRF) es una vulnerabilidad de seguridad en la que un atacante puede forzar a un servidor web para que realice solicitudes HTTP en su nombre.

En un ataque de SSRF, el atacante utiliza una entrada del usuario, como una URL o un campo de formulario, para enviar una solicitud HTTP a un servidor web. El atacante manipula la solicitud para que se dirija a un servidor vulnerable o a una red interna a la que el servidor web tiene acceso.

El ataque de SSRF puede permitir al atacante acceder a información confidencial, como contraseñas, claves de API y otros datos sensibles, y también puede llegar a permitir al atacante (en función del escenario) ejecutar comandos en el servidor web o en otros servidores en la red interna.

Una de las diferencias clave entre el SSRF y el CSRF es que el SSRF se ejecuta en el servidor web en lugar del navegador del usuario. El atacante no necesita engañar a un usuario legítimo para hacer clic en un enlace malicioso, ya que puede enviar la solicitud HTTP directamente al servidor web desde una fuente externa.

Para prevenir los ataques de SSRF, es importante que los desarrolladores de aplicaciones web validen y filtren adecuadamente la entrada del usuario y limiten el acceso del servidor web a los recursos de la red interna. Además, los servidores web deben ser configurados para limitar el acceso a los recursos sensibles y restringir el acceso de los usuarios no autorizados.

En esta clase, estaremos utilizando Docker para crear redes personalizadas en las que podremos simular un escenario de red interna. En esta red interna, intentaremos a través del SSRF apuntar a un recurso existente que no es accesible externamente, lo que nos permitirá explorar y comprender mejor la explotación de esta vulnerabilidad.

Para crear una nueva red en Docker, podemos utilizar el siguiente comando:

➜ docker network create --subnet= <nombre_de_red>

Donde:

- subnet: es la dirección IP de la subred de la red que estamos creando. Es importante tener en cuenta que esta dirección IP debe ser única y no debe entrar en conflicto con otras redes o subredes existentes en nuestro sistema.
- nombre_de_red: es el nombre que le damos a la red que estamos creando.

Además de los campos mencionados anteriormente, también podemos utilizar la opción ‘–driver‘ en el comando ‘docker network create’ para especificar el controlador de red que deseamos utilizar.

Por ejemplo, si queremos crear una red de tipo “bridge“, podemos utilizar el siguiente comando:

➜ docker network create --subnet= --driver=bridge <nombre_de_red>

En este caso, estamos utilizando la opción ‘–driver=bridge‘ para indicar que deseamos crear una red de tipo “bridge“. La opción –driver nos permite especificar el controlador de red que deseamos utilizar, que puede ser “bridge“, “overlay“, “macvlan“, “ipvlan” u otro controlador compatible con Docker.

README.md principal

README1.md <--> README3.md