VÍDEO intypedia007es

LECCIÓN 7: SEGURIDAD EN APLICACIONES WEB. INTRODUCCIÓN A LAS TÉCNICAS DE INYECCIÓN SQL

AUTOR: Chema Alonso

Consultor de Seguridad en Informática 64. Microsoft MVP Enterprise Security

BERNARDO

Hola, bienvenido a intypedia. Seguramente hayas podido leer recientemente en algún medio de comunicación que un atacante ha sido capaz de robar los datos de los clientes de una empresa a través de su página web y te hayas preguntado cómo lo han hecho. En esta lección vamos a intentar explicar brevemente cómo puede ser esto posible. ¡Acompáñanos!

ESCENA 1. ARQUITECTURA BÁSICA DE UNA APLICACIÓN WEB. INCIDENTES DE SEGURIDAD

ALICIA

Muchos internautas se preguntarán cómo es posible que un atacante pueda tener acceso a información sensible de una empresa, que no está publicada en Internet, a través de su página web, por ejemplo su base de datos o modificar los valores mostrados en ella. La respuesta técnica es mucho más sencilla de lo que se puede suponer a priori.

BERNARDO

Efectivamente. Aunque el servidor de bases de datos donde se almacena la información de la empresa se encuentre detrás de una serie de firewalls u otros mecanismos de protección, existe un canal de comunicaciones entre el servidor web y el servidor de bases de datos que es

lo que el atacante utiliza para llegar hasta los datos guardados. Alicia, ¿nos describes un poco la arquitectura básica de una aplicación web que utiliza una base de datos?

ALICIA

Por supuesto, vamos a ello. Aunque una aplicación web puede ser tan compleja como necesite el sistema, las aplicaciones típicas tienen la siguiente estructura. En ella se puede ver que hay tres niveles claramente diferenciados por sus funciones.

El primero de ellos es el encargado del interfaz de usuario, es decir, la parte que gestiona la interacción Humano-Sistema, y que en una aplicación web típica es el navegador de Internet.

El segundo de los niveles es donde reside la lógica de la aplicación. Esta correrá sobre los servidores web y los servidores de aplicaciones, que haciendo uso de diferentes tecnologías podrán generar conocimiento o procesar información con un fin concreto.

Por último tendríamos el almacén de datos, es decir, el repositorio de la información donde se guarda el conocimiento de una organización. Este repositorio puede estar implementado por un árbol LDAP, una base de datos relacional, un almacén con datos en XML o un simple fichero de datos.

Existe una conexión entre cada uno de los niveles adyacentes. El usuario, a través de la información que envía por el navegador de Internet, interactúa con el servidor web y, a su vez, la lógica de la aplicación interactúa con el almacén de datos para leer y escribir información en el sistema.

Por supuesto, cada interacción tiene sus conjuntos de protocolos y lenguajes. Así, entre el nivel de interfaz y el nivel de lógica de aplicación se pueden utilizar protocolos como http o https para enviar la información, mientras que entre el servidor web y los repositorios de datos se lanzarán consultas en lenguajes propios sobre protocolos de red creados específicamente para esta función.

Por ejemplo, si el repositorio es un árbol LDAP, la lógica de la aplicación lanzará consultas LDAP con filtros de búsqueda LDAP. Si fuera una base de datos relacional se utilizará consultas en lenguaje SQL sobre protocolos como Tabular Data Stream u Oracle Net, y en el caso de un repositorio en formato XML se utilizarán consultas XPath.

BERNARDO

¡Qué interesante! Por tanto, ante la pregunta ¿cómo es posible que un atacante extraiga información de un sistema informático con una aplicación web?, la respuesta parece bastante clara. Lo más probable es que haya un fallo de seguridad en el código de esa aplicación web.

Actualmente, según la conocida organización OWASP (Open Web Application Security Project) los 10 vectores de ataque más probables a estas aplicaciones son: *A1: Injection, A2: Cross-Site Scripting (XSS), A3: Broken Authentication and Session Management, A4: Insecure Direct Object References, A5: Cross-Site Request Forgery (CSRF), A6: Security Misconfiguration, A7: Insecure Cryptographic Storage, A8: Failure to Restrict URL Access, A9: Insufficient Transport Layer Protection, A10: Unvalidated Redirects and Forwards*. Estos ataques intentarán explotar fallos

de seguridad en aplicaciones web (en tecnologías de cliente o de servidor) con algún fin concreto, por ejemplo, extraer información de una base de datos. Tiempo tendremos de profundizar en todos ellos. A continuación, vamos a destacar los fundamentos de uno de los ataques más famosos y efectivos: la inyección de código SQL.

ESCENA 2. INYECCIÓN DE CÓDIGO EN APLICACIONES WEB

ALICIA

Uno de los principales problemas de seguridad al que se enfrentan las aplicaciones web es el de protegerse contra los ataques de inyección. Bernardo, ¿nos cuentas en qué consiste el fallo utilizado por los atacantes en este tipo de vulnerabilidades?

BERNARDO

Claro Alicia, es sencillo de explicar. La vulnerabilidad que aprovecha el atacante es que la aplicación web no valida correctamente los datos que introduce el usuario en el interfaz y que se utilizan a posteriori en consultas al repositorio de datos. Veámoslo mejor con un ejemplo.

Supongamos que una aplicación web tiene un sistema de validación de usuarios por medio de un formulario en el que se debe introducir un código de usuario y una contraseña. Si el programador no valida correctamente la información introducida y concatena los datos directamente, construyendo la consulta que se envía al servidor de la base datos, el atacante puede interactuar directamente con la base de datos, por ejemplo con consultas SQL, y por tanto dependiendo del entorno, extraer, modificar o borrar información.

ALICIA

Como se puede ver, el atacante no ha introducido una contraseña válida sino que ha introducido una condición, usando el carácter comilla, que al ser concatenada en la consulta SQL que se va a enviar a la base de datos hace que siempre se cumpla, con lo que la lógica de la aplicación recibe de la aplicación un usuario autenticado.

Algún internauta podría pensar que se trata únicamente de utilizar de manera efectiva el carácter comilla, pero no es así. La vulnerabilidad consiste en no filtrar los datos correctamente. Si en lugar de un formulario de login con campos de texto el atacante aprovechara la llamada a un programa con un parámetro numérico, le sería suficiente con poner un espacio entre el número y el comando que quiere inyectar para conseguir ejecutar comandos en la base de datos relacional.

Por ejemplo, algo tan sencillo como http://www.server.com/app.aps?id=1; shutdown – podría ser un ataque que parara el motor de base de datos mediante SQL Injection y no es necesario introducir ningún carácter comilla.

BERNARDO

Cierto y esto no es sólo para las bases de datos. El ataque se puede realizar con éxito en sistemas con cualquier tipo de repositorio de datos. Sólo se deben cambiar los tipos de comandos a inyectar y se podrá ejecutar casi cualquier comando que permita el motor.

ALICIA

Por supuesto. Existen ciertas limitaciones a las que se enfrenta el atacante como son los privilegios que tenga la cuenta de la aplicación web dentro del motor de repositorio de datos o la información que muestra la aplicación una vez que se han ejecutado los comandos. Es por ello que los investigadores han desarrollado técnicas muy afinadas para la extracción de la información que almacena el sistema, incluso sin llegar a ver nunca esos datos directamente en la web.

BERNARDO

Sí, a la hora de extraer los datos del motor se diferencian entre ataques inbound y ataques outbound. Los primeros son aquellos en los que la propia aplicación web, manipulándola adecuadamente, muestra la información que el atacante extrae. Por el contrario, los ataques outbound extraen la información mediante la generación de mensajes de error o el desencadenamiento de acontecimientos que les permiten extraer los datos. Hay pruebas de concepto en los que los datos se extraen generando consultas DNS a un servidor controlado en el que se deja en el log los datos extraídos.

ALICIA

Además, tampoco le es necesario al atacante visualizar los datos, como en los ataques inbound u outbound, que extrae de una base de datos a través de la aplicación web ya que existen los llamados ataques a ciegas. En éstos los datos se infieren por el comportamiento que tiene una aplicación web tras la inyección de un determinado comando, tal y como va explicarnos Bernardo a continuación.

ESCENA 3. TÉCNICAS DE INYECCIÓN A CIEGAS

BERNARDO

Como bien ha dicho Alicia, en el momento que un atacante puede inyectar algo de código en las consultas que se lanzan contra las bases de datos, puede forzar cambios en el comportamiento de la aplicación en función de distintas inyecciones. Si un atacante es capaz de encontrar un comportamiento constante en inyecciones de código con todos los condicionantes de la consulta generada con valores a True, es decir, que se cumplen todas las condiciones, o un comportamiento constante en inyecciones de código con algún valor a False, es decir, que no se cumple dicho condicionante, entonces el atacante puede construir una lógica binaria para extraer toda la información de la base de datos.

ALICIA

Para entender esto mejor vamos a suponer un entorno sencillo. Supongamos que un programador ha construido una página web en la que se muestra una noticia. Esa noticia está almacenada en una tabla de una base de datos relacional a la que el programador accede con una sencilla consulta del tipo:

Select * from noticias where id='+noticia_id+'

El desarrollador utiliza el valor noticia_id para seleccionar la noticia a mostrar y éste es recogido por un parámetro GET desde el cliente, algo como:

www.servidor.com/mostrar_noticias.php?noticia_id=1

Si la construcción, supongamos, de la consulta SQL con el valor noticia_id se realiza concatenando una cadena de texto y sin filtrar correctamente el valor que se envía por noticia_id, entonces tendremos una vulnerabilidad de SQL Injection.

BERNARDO

Con esa vulnerabilidad el atacante podría probar, por ejemplo, el comportamiento de la aplicación ante situaciones extremas con condiciones a True y condiciones a False, para establecer una lógica binaria.

Así, podría inyectar algo como *http://www.servidor.com/mostrar_noticias.php?noticia_id=1* and 1=1 y *http://www.servidor.com/mostrar_noticias.php?noticia_id=1* and 1=0. En el primer caso, la consulta SQL resultante no cambia el funcionamiento de la aplicación, pero en el segundo caso, hace que la consulta SQL no devuelva ningún resultado. Esto generaría dos consultas SQL tales como:

Select campos from tablas where condiciones and 1=1

-

Select campos from tablas where condiciones and 1=0

Si ante estas inyecciones la aplicación muestra diferentes comportamientos, por ejemplo en el primer caso se muestra una noticia y en el segundo se muestra un mensaje de error, entonces se podría extraer la información haciendo un ataque a ciegas.

ALICIA

Con estos comportamientos el atacante sabría que si se muestra la noticia ante una inyección, es que ha puesto una condición que da True, mientras que si se muestra el código de error, es que ha puesto una condición que da False. Por ello, para extraer la información podría hacer inyecciones del tipo siguiente:

http://www.servidor.com/mostrar_noticias.php?noticia_id=1 and (100=(select top 1 ascii(substring(login,1,1)) from usuarios))

En la que se está preguntando si el valor ASCII de la primera letra del nombre del primer usuario es igual a 100, es decir, si el nombre del usuario comienza por d minúscula. Utilizando este tipo de consultas se puede volcar toda la base de datos sin ni tan siquiera ver la

información, lógicamente también es posible “adivinar” los nombres de las tablas. Destacar que esta técnica concreta genera un número considerable de peticiones a la base de datos que podrían ser registradas en un log o levantar las sospechas de un IDS.

BERNARDO

Estas técnicas a ciegas han evolucionado analizando los comportamientos de las aplicaciones y buscando comportamientos distintos en todas las características de las páginas de respuesta, llegando hasta a utilizar retardos de tiempo para reconocer los valores True y False.

ESCENA 4. EVITANDO LAS TÉCNICAS DE INYECCIÓN. MITIGACIÓN

ALICIA

Como hemos visto, si la aplicación no filtra correctamente los parámetros que vienen desde el usuario para generar la consulta, el atacante puede buscar la manera de extraer la información del repositorio de datos utilizando consultas más o menos complejas.

BERNARDO

Para ello es necesario hacer especial hincapié en el filtrado de los datos que vienen desde el cliente. En los entornos de desarrollo existen componentes avanzados para la construcción segura de consultas de extracción de datos, por lo que es recomendable utilizar este tipo de componentes para crear estas consultas y no hacerlo manualmente mediante concatenación de cadenas de texto y variables.

ALICIA

Además, algunos de estos IDEs vienen con analizadores de código estático que son capaces de detectar este tipo de vulnerabilidades mediante un exhaustivo análisis del código fuente.

BERNARDO

Bueno, hasta aquí la lección de hoy. En la página Web de intypedia encontrarás documentación adicional a esta lección. ¡Adiós!

ALICIA

Hasta la próxima.

Guión adaptado al formato intypedia a partir del documento entregado por D. José María Alonso Cebrián

Madrid, España, mayo de 2011

http://www.intypedia.com

http://twitter.com/intypedia