Recomendaciones de **OWASP** sobre **seguridad**
en el desarrollo de **Aplicaciones Web**
Guías de Referencias Comentadas

Guía de Referencias sobre
Autentificación

Traducida y comentada por
Eugenia Bahit

The Original Hacker N°14
Serie: Guías Comentadas

Guía de Referencias sobre Autentificación – Comentada por Eugenia Bahit

Revisado el **2 de agosto de 2015**

Contenido

Introducción

La **autentificación** es el proceso de verificar que un individuo, entidad o sitio Web es quien dice ser. En el contexto de una aplicación Web, la autentificación, comúnmente es realizada mediante el envío de un nombre de usuario o ID y, uno o más datos de información privada que solo un determinado usuario debe conocer.

El **manejo de sesiones** es un proceso por el cual un servidor mantiene el estado de una entidad interactuando con él. Esto es requerido por un servidor para recordar como debe reaccionar a las peticiones posteriores a lo largo de una transacción. Las sesiones son mantenidas en el servidor por un identificador de sesión el cual puede ser pasado y devuelto entre el cliente y el servidor al transmitir y recibir solicitudes. Las sesiones deben ser únicas por usuario e informáticamente, muy difíciles de predecir.

Reglas generales de autentificación

ID de usuario

Asegúrese de que sus nombres/identificadores de usuarios no sean sensibles a mayúsculas y minúsculas. El usuario 'smith' y el usuario 'Smith' deberían ser el mismo usuario.

> En este punto en concreto, la no distinción entre mayúsculas y minúsculas debería considerarse un vulnerabilidad puesto que la equidad automatizada entre 'smith' y 'Smith' facilitaría el reconocimiento de un nombre de usuario a un atacante.
> Un ejemplo que puede ayudar a entender esto como una vulnerabilidad sería el siguiente:
>
> Dado un usuario `aDMin` cuya contraseña es `123456`, cualquier atacante que intentase la tan conocida (y para nada segura) combinación de usuario y contraseña `admin/123456`, podría acceder al sistema con menos intentos que si tuviese que acertar la combinación interna de mayúsculas y minúsculas en el propio nombre de

> usuario, puesto que matemáticamente, se requiere de un mayor número de combinaciones posibles.

Dirección de correo electrónico como ID de usuario

Muchos sitios utilizan la dirección de correo electrónico como identificador de usuario, lo cual es un buen mecanismo para asegurar un identificador único por cada usuario sin agregarle a éstos la carga de tener que recordar un nuevo nombre de usuario. Sin embargo, muchas aplicaciones Web no tratan correctamente las direcciones de correo electrónico, debido a conceptos equivocados sobre lo que constituye una dirección de correo electrónico válida.

En concreto, es completamente válido tener una dirección correo electrónico que:

- Es sensible a mayúsculas y minúsculas en la parte local
- Tiene caracteres no alfanuméricos en la parte local (incluyendo + y **@**)
- Tiene cero o más etiquetas (aunque ciertamente cero no va a ocurrir)

La parte local es la parte de la dirección de correo electrónico que se encuentra a la izquierda del caracter '@'. El dominio es la parte de la dirección de correo electrónico que se encuentra a la derecha del caracter '@' y consiste en cero o más etiquetas unidas por el caracter de punto.

Al momento de estar escribir este artículo, el RFC 5321 es el estándar actual que define el protocolo SMTP y lo que constituye una dirección de correo electrónico válida.

Por favor, tenga en cuenta que las direcciones de correo electrónico deberían ser consideradas datos públicos. En aplicaciones de alta seguridad, podrían asignarse los nombres de usuario y ser secretos en lugar de ser datos públicos definidos por el usuario.

> **Este último aspecto es de vital importancia y el programador no debe dejar de**

> **tenerlo en cuenta ni mucho menos, descartarlo.**
> Aplicaciones Web destinadas al manejo de información sensible, tales como cuestiones relacionadas al dinero o datos privados que puedan comprometer la seguridad física de las personas o sus bienes, no deberían utilizar jamás una dirección de correo electrónico como identificador de usuario. En su defecto, debería ser el sistema quien generase dicha información de forma automática y la provea al usuario.
>
> Otro factor a tener en cuenta, es proteger por diseño dicha información para evitar una violación de privacidad indirecta que ponga en riesgo la seguridad de la aplicación. Es el caso de los nombres de usuario que, por diseño, se utilizarán como identificadores públicos del usuario, frente a otros usuarios del sistema. Imagines un foro como StackOverflow donde la dirección de correo electrónico no solo fuese el identificador de usuario para el proceso de autentificación, sino que por diseño, se lo utilizase para que los usuarios se identificasen entre sí. En estos casos, sería preferible utilizar otro identificador "menos comprometido" para que los usuarios se identificaran entre sí.

Validación

Muchas aplicaciones Web contienen expresiones regulares informáticamente muy costosas e inexactas para intentar validar las direcciones de correo electrónico.

Cambios recientes generaron que el número de falsos negativos se viera incrementado, particularmente debido a:

- El aumento de popularidad de las sub-direcciones de proveedores como Gmail (comúnmente usando + como token en la parte local para afectar la entrega)
- Nuevos gTLDs con nombres largos (muchas expresiones regulares comprueban el número y longitud de cada etiqueta en el dominio)

Siguiendo el RFC 5321, las mejores prácticas para la validación de una dirección de correo electrónico deberían ser:

- Comprobar la presencia de al menos un símbolo de @ en la dirección
- Asegurarse de que la parte local no es de más de 64 bytes
- Asegurarse de que el dominio no es de más de 255 bytes
- Asegurarse que sea una dirección de entrega verídica (NdT: se refiere a que el correo pueda ser entregado)

Para asegurarse que una dirección de entrega sea verídica, la única forma es enviar un correo electrónico al usuario y que éste deba tomar alguna acción para confirmar que lo ha recibido. Más allá de confirmar que la dirección de correo electrónico es válida y reciba los mensajes, esto también proporciona una confirmación positiva de que el usuario tiene acceso al buzón de correo y es probable que esté autorizado a usarlo. Esto no significa que otros usuarios no tengan acceso al mismo buzón de correo, cuando por ejemplo el usuario utiliza un servicio que genera una dirección de correo electrónico desechable.

> Las validaciones de correo electrónico pueden llegar a ser muy costosas para la aplicación y para el servidor.
> Tal y como asegura la guía de OWASP, es necesario asegurarse de que la dirección de correo exista (sea verídica) y para ello, nunca deberá enviarse el correo de comprobación SIN ANTES haber validado la entrada del usuario, ya que direcciones de correo electrónico mal formadas, tales como 'foobar', podrían generar extensas colas de salida en el servidor, que provocasen un consumo innecesario de recursos.
> Por otra parte, los correos electrónicos no confirmados, deberían persistir SOLO de forma temporal en el sistema, para evitar que el colapso de una base de datos genere la falta de disponibilidad de la aplicación Web. Esto demanda la toma de medidas de "limpieza" automatizada en el sistema.
> Un ejemplo de persistencia temporal y acciones de confirmación, puede obtenerse en el artículo **«Emulación de tokens de seguridad temporales para el registro de usuarios»** el cual puede descargarse desde la siguiente URL:
> http://library.originalhacker.org/biblioteca/articulo/ver/115

Normalización

Como la parte local de las direcciones de correo electrónico son, de hecho, sensibles a mayúsculas y minúsculas, es importante almacenar y comparar las direcciones de correo electrónico correctamente. Para normalizar la entrada de una dirección de correo electrónico, debería convertir la parte del dominio SOLO a minúsculas.

Desafortunadamente, esto hace y hará a la entrada, más difícil de normalizar y de coincidir correctamente con los intentos del usuario.

Es razonable aceptar solo una única capitalización de diferentes alternativas para direcciones de correo electrónico idénticas. Sin embargo, en este caso es crítico para:

- Almacenar la parte del usuario tal y como fue provista y verificada por el usuario en el proceso de verificación
- Realizar comparaciones `lowercase(provista) == lowercase(almacenada)`

Implementar controles adecuados de fortaleza de contraseña

Una de las principales preocupaciones cuando se utilizan contraseñas para la autentificación, es la fortaleza de las contraseñas. Una política de contraseñas "fuertes" hace que sea difícil o incluso improbable adivinar la contraseña a través de medios manuales o automatizados. Las siguientes características definen una contraseña fuerte:

Advertencia

Las siguientes indicaciones están disputadas. Por favor, vea la presentación de OWASP (en inglés), "Your Password Complexity Requirements are Worthless - OWASP AppSecUSA

2014" para más información.

Longitud de la contraseña

Las contraseñas más largas proporcionan una mayor combinación de caracteres y en consecuencia hacen que sea más difícil de adivinar para un atacante.

- La longitud **mínima** de las contraseñas debería ser **forzada** por la aplicación.
    - Las contraseñas **menores a 10 caracteres** son consideradas débiles ([1]).

Mientras que la longitud mínima forzada puede causar problemas para la memorización de la contraseña en algunos usuarios, las aplicaciones deberían alentarlos a establecer *frases de paso o passphrases* (frases o combinaciones de palabras) que pueden ser mucho más largas que las contraseñas típicas y mucho más fáciles de recordar.

- La longitud **máxima** de la contraseña no debería establecerse **demasiado baja**, ya que evitará que los usuarios puedan crear frases de paso (passphrases). La longitud máxima típica es de 128 caracteres.
    - Frases de paso de menos de 20 caracteres usualmente son consideradas débiles si solo se emplean letras minúsculas.

> Es muy comprensible que esta sección haya presentado disputas en el propio seno de OWASP, ya que la longitud de una contraseña, es un hecho que no influye demasiado en los intentos de 'adivinación' manuales.
> Las contraseñas se obtienen mayormente con mejores resultados, mediante el estudio y perfilado del sujeto. De hecho, la mayoría de las veces lleva menos tiempo la obtención manual de una contraseña que la automatizada a través de fuerza bruta.
> Por ejemplo, una típica mamá que tiene su escritorio empapelado de fotos de sus tres hijos, es más probable que utilice contraseñas largas que cortas, ya que muy probablemente utilice una contraseña formada por los nombres de sus tres hijos o sus

> fechas de nacimiento ordenadas desde el primogénito hacia el más pequeño.

Complejidad de la contraseña

Las aplicaciones deberían imponer reglas de complejidad de contraseñas para evitar las contraseñas fáciles de adivinar. Los mecanismos de contraseñas deberían permitir al usuario, poder tipear casi cualquier caracter como parte de su contraseña, incluyendo el caracter de espacio. La contraseñas deberían, obviamente, ser sensibles a mayúsculas y minúsculas a fin de incrementar la complejidad de las mismas. Ocasionalmente, encontramos sistemas donde las contraseñas no son sensibles a mayúsculas y minúsculas, frecuentemente debido a problemas de sistemas heredados como los viejos ordenadores centrales que no tenían contraseñas sensibles a mayúsculas y minúsculas.

El mecanismo de cambio de contraseña debería requerir un nivel mínimo de complejidad que tenga sentido para la aplicación y su población de usuarios. Por ejemplo:

- La contraseña debe reunir al menos 3 de las siguientes 4 reglas de complejidad
    - al menos 1 mayúscula (A-Z)
    - al menos 1 minúscula (a-z)
    - al menos 1 dígito (0-9)
    - al menos 1 caracter especial (puntuación) — no olvidar de tratar también, a los espacios en blanco como un caracter especial
- al menos 10 caracteres
- no más de 128 caracteres
- no más de 2 caracteres idénticos consecutivos (ej., 111 no está permitido)

> Desde un punto de vista lógico-matemático, la complejidad de una contraseña la hace tan difícil de acertar automáticamente como desde el punto de vista psicológico, para adivinarla de forma manual. Por ello, independientemente de la opinión personal de

> cada uno, poner mayor énfasis en la complejidad de la contraseña que en su longitud, con certeza la hará mucho menos probable de vulnerar.

Topologías de contraseña

- Prohibir topologías de contraseñas de uso común
- Forzar a varios usuarios a utilizar diferentes topologías de contraseña
- Exigir un cambio mínimo de topología entre viejas y nuevas contraseñas

> En lo particular, se debería tener especial cuidado con este aspecto ya que para ponerlo en práctica, supone el almacenamiento histórico de las contraseñas de un usuario, algo que puede generar desconfianza en los usuarios ya que se encuentran frente a un sistema informático que sabe más sobre ellos que ellos mismos.
> Desde mi punto de vista, deberían solo contrastarse la última contraseña con la nueva, para así evitar una excesiva intromisión en la privacidad del usuario. Téngase en cuenta que el historial de contraseñas elegidas por un usuario, en términos criminológicos, puede ser utilizado para complementar el perfil de éste.

Información adicional

- Asegúrese de que todos los caracteres que el usuario escribe están realmente incluidos en la contraseña. Hemos visto sistemas que truncan la contraseña a una longitud inferior de la que el usuario provee (ej., truncada a los 15 caracteres cuando se han ingresado 20).
    - Esto es manejado usualmente al establecer la longitud de TODOS los campos de contraseña exactamente como la longitud máxima de la contraseña. Esto es particularmente importante si su longitud máxima de contraseña es corta, como 20-30 caracteres.

Si la aplicación requiere políticas de contraseña más complejas, será necesario ser muy

claro sobre cuáles son esas políticas.

- La política requerida necesita ser indicada explícitamente en la página de cambio de contraseña
    - asegúrese de enumerar cada caracter especial que permite, para que sea evidente para el usuario

Recomendación:

- Lo ideal, sería que la aplicación indicara al usuario cómo escribir su nueva contraseña y cuánto de la directiva de complejidad de su nueva contraseña cumple
    - De hecho, el botón de envío debería verse atenuado hasta que la nueva contraseña reúna los requisitos establecidos en la política de complejidad de contraseña y la segunda copia de la nueva contraseña coincida con la primera. Esto hará que sea mucho más fácil, para el usuario, entender la política de complejidad y cumplirla.

Independientemente de cómo se comporte la UI, cuando un usuario envía su solicitud de cambio de contraseña:

- Si la nueva contraseña no cumple con la política de complejidad de contraseña, el mensaje de error debería describir TODAS las reglas de complejidad con las cuáles la nueva contraseña no cumple y no sola la primera regla con la que no cumpla.

Implementar un mecanismo seguro de recuperación de contraseña

Es común que una aplicación tenga un mecanismo que provea al usuario un medio para acceder a su cuenta en caso de que olvide su contraseña. Por favor, para más detalles sobre esta característica, vea Forgot Password Cheat Sheet (en inglés).

Almacenar contraseñas de forma segura

Es fundamental para una aplicación, almacenar contraseñas usando la técnica criptográfica correcta. Para conocer más sobre este mecanismo, vea Password Storage Cheat Sheet (en inglés).

Transmitir contraseñas sólo sobre TLS u otro transporte fuerte

Ver: Transport Layer Protection Cheat Sheet (en inglés)

> Al hablar de TLS nos estamos refiriendo al sucesor de SSL

La página de inicio de sesión y todas las páginas autentificadas subsiguientes, deberían ser accedidas exclusivamente sobre TLS u otro transporte fuerte. La página de inicio de sesión principal, conocida como "landing page", debe ser servida sobre TLS u otro transporte fuerte.

Si no se utiliza TLS u otro transporte fuerte para la landing page de inicio de sesión, se permite a un atacante modificar el *action* del formulario de inicio de sesión, generando que las credenciales del usuario sean enviadas a una ubicación arbitraria.

Si no se utiliza TLS u otro transporte fuerte para las páginas autentificadas que se habilitan luego del inicio de sesión, un atacante puede ver la ID de sesión sin cifrar y comprometer la sesión autentificada del usuario.

> Muchos programadores, sobre todo los acostumbrados a desarrollar en ordenadores con sistemas operativos de uso personal, suelen tenerle un poco de 'miedo' a la instalación de certificados de seguridad y los mecanismos de transporte de información segura como TLS (antiguamente conocido como SSL).
> Sin embargo, implementar dichos mecanismos no es complejo en lo absoluto. En https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-with-a-free-signed-ssl-certificate-on-a-vps puede obtenerse un tutorial de cómo hacer esto.

Solicitar volver a autentificarse para funciones sensibles

Con el fin de mitigar ataques CSRF y de secuestro de sesión (hijacking), es importante solicitar las credenciales actuales de una cuenta en los siguientes casos:

- Antes de modificar información sensible (como la contraseña del usuario, la dirección de correo electrónico del usuario)
- Antes de transacciones sensibles (como enviar una compra a una nueva dirección).

Sin esta contramedida, un atacante puede ser capaz de ejecutar transacciones sensibles a través de un ataques CSRF o XSS sin necesidad de conocer las credenciales actuales del usuario. Adicionalmente, un atacante puede obtener, temporalmente, acceso físico al navegador del usuario o robar su ID de sesión para tomar el control de la sesión del usuario.

> A veces lo menos pensado por su simpleza es lo que mayor daño podría causar. Muchos programadores evitan este tipo de medidas argumentando que “solo 'Fulanito' tendrá acceso al sistema”. Pero 'Fulanito' podrá tener acceso como administrador. Y 'Fulanito' podrá acceder desde su ordenador y descuidarlo un solo minuto. Y en ese minuto, cualquier persona -y no necesariamente mal intencionada,

> basta con que sea una persona 'metida' y poco cuidadosa- podría tomar posición de su ordenador y 'tocar lo que no debe'. Si es una funcionalidad sensible, no importa quién hará uso de ella. **Siempre habrá que tomar medidas preventivas.**

Utilizar la autentificación por múltiples factores

La autentificación por múltiples factores (MFA por las siglas en ingles de "Multi-factor authentication") es el uso de más de un factor de autentificación para iniciar sesión o procesar una transacción, mediante:

- Algo que se conoce (detalles de la cuenta o contraseñas)
- Algo que se tiene (tokens o teléfonos móviles)
- Algo que se es (factores biométricos)

Los esquemas de autentificación como las contraseñas de un solo uso (OTP por las siglas en inglés de "One Time Passwords") implementadas utilizando un token físico (hardware) también pueden ser un factor clave en la lucha contra ataques tales como los ataques CSRF y malware del lado del cliente. Un considerable número de los token de hardware para MFA disponibles en el mercado, permiten una buena integración con las aplicaciones Web. Ver: [2] (en inglés).

> Los token de seguridad por hardware no suelen ser tan conocidos en América Latina y países subdesarrollados como lo son en países del primer mundo.
> En esta zona del mundo, los dispositivos de seguridad por hardware suelen ser económicamente muy costosos y no demasiado simples de conseguir, por lo que su implementación debería ser cuidadosamente estudiada antes de tomar desiciones.

Autentificación TLS

La autentificación TLS, también conocida como autentificación TLS mutua, consiste en

que ambos, navegador y servidor, envíen sus respectivos certificados TLS durante el proceso de negociación TLS (*handshaking*). Así como se puede validar la autenticidad de un servidor mediante el certificado y, preguntar a una Autoridad de Certificación conocida (CA, por las siglas en inglés de "Certificate Authority") si la certificación es válida, el servidor puede autentificar al usuario recibiendo un certificado desde el cliente y validándolo contra una CA o su propia CA. Para hacer esto, el servidor debe proveer al usuario de un certificado generado específicamente para él, asignando valores que puedan ser usados para determinar que el usuario debe validar el certificado. El usuario instala los certificados en el navegador y los usa para el sitio Web.

Es una buena idea hacer esto cuando:

- Es aceptable (o incluso preferido) que el usuario sólo tenga acceso a la página web desde una sola computadora/navegador.
- El usuario no se asusta fácilmente por el proceso de instalación de certificados TLS en su navegador o habrá alguien, probablemente de soporte de TI, que hará esto para el usuario.
- El sitio web requiere un paso adicional de seguridad.
- El sitio Web es de la intranet de una compañía, empresa u organización.

Por lo general, no es una buena idea utilizar este método para la mayor parte de los sitios Web de acceso público que tendrán un usuario promedio. Por ejemplo, no será una buena idea implementar esto en un sitio Web como Facebook. Si bien esta técnica puede evitar que el usuario tenga que escribir una contraseña (protegiéndola así contra el robo desde un keylogger promedio), aún se considera una buena idea emplear el uso de una contraseña combinada con la autentificación TLS.

Para más información, ver: Client-authenticated TLS handshake

> Definitivamente, este mecanismo de autentificación, es el más reticente para los

> usuarios y desde mi punto de vista, solo debería ser implementado en una Intranet (donde exista personas de soporte IT que se encargue de la instalación de los certificados).

Autentificación y mensajes de error

En el caso de las funcionalidades de autentificación, los mensajes de error implementados de forma incorrecta pueden ser utilizados con el propósito de obtener y almacenar identificadores de usuario y contraseñas. Una aplicación, debería responder (tanto en los encabezados HTTP como en el contenido HTML) de forma genérica.

Respuestas de autentificación

Una aplicación debería responder mensajes de error genéricos independientemente de si era incorrecto el identificador de usuario o la contraseña. Tampoco debería dar información sobre el estado de una cuenta existente.

Ejemplo de respuestas incorrectas

- "Inicio de sesión para el usuario foo: contraseña incorrecta"
- "Falló el inicio de sesión: usuario no válido"
- "Falló el inicio de sesión: cuenta deshabilitada"
- "Falló el inicio de sesión: usuario inactivo"

Ejemplo de respuestas correctas

- "Falló el inicio de sesión: Usuario o contraseña incorrectos"

La respuesta correcta no debería indicar si el identificador de usuario o la contraseña es

el parámetro incorrecto y por lo tanto, inferir un identificador de usuario válido.

Códigos de error y URLs

La aplicación puede retornar un código de error HTTP diferente dependiendo del resultado del intento de autentificación. Puede responder con un 200 para un resultado positivo y con un 403 para un resultado negativo. Aunque una página de error genérico sea mostrada al usuario, el código de respuesta HTTP puede ser diferente, permitiendo filtrar la información sobre si la cuenta es válida o no.

Prevenir ataques por fuerza bruta

Si un atacante es capaz de adivinar una contraseña sin ser deshabilitada debido a intentos de autentificación fallidos, el atacante tiene la oportunidad de continuar con un ataque de fuerza bruta hasta que la cuenta se vea comprometida. La automatización de los ataques de fuerza bruta para adivinar contraseñas en aplicaciones Web son un desafío muy usual.

Los mecanismos de bloqueo de contraseña deberían ser empleados para bloquear una cuenta si se realiza más de un número predeterminado de intentos fallidos de autentificación.

Los mecanismos de bloqueo de contraseña tienen una debilidad lógica. Un atacante que emprende un gran número de intentos de autentificación sobre nombres de cuentas conocidas puede producir como resultado, el bloqueo de bloques enteros de cuentas de usuario. Teniendo en cuenta que la intención de un sistema de bloqueo de contraseña es proteger de ataques por fuerza bruta, una estrategia sensata es bloquear las cuentas por un período de tiempo (ej., 20 minutos). Esto ralentiza considerablemente a los atacantes mientras que permite automáticamente, reabrir las cuentas para los usuarios legítimos.

Además, la autenticación de múltiples factores es un muy poderoso elemento de

disuasión cuando se trata de prevenir los ataques de fuerza bruta ya que las credenciales son un blanco móvil. Cuando la autenticación de múltiples factores se implementa y activa, el bloqueo de cuentas ya no es necesario.

> A veces cuesta un poco entender la debilidad y objetivo del bloqueo de contraseñas, por eso lo explicaré de un modo simple.
>
> **¿Para qué bloquear una cuenta?** Si tras varios intentos de autentificación fallidos, la contraseña no es bloqueada, tras cada intento el atacante gana una posibilidad más de adivinar la clave y tener éxito. Se bloquea la contraseña para que el atacante deje de intentar adivinar la clave y fracase.
>
> Pero **¿qué sucede si se bloquea la cuenta de forma permanente?** Este puede convertirse en el objetivo real de un atacante. Por ejemplo, supongamos que se establece un máximo de 3 intentos fallidos. Entonces el atacante, a propósito, falla 3 veces la autentificación de cada uno de los usuarios. Por consiguiente, bloqueará todas las cuentas del sistema y en un escenario ideal (para el atacante), el sistema quedaría en desuso ya que todos los accesos estarían bloqueados.
>
> **¿Para qué se bloquea entonces de forma temporal?** Para que los legítimos usuarios puedan acceder y el sistema no quede completamente bloqueado.
>
> Pero la pregunta debería ser si **¿es realmente útil el bloqueo de cuentas?** Por lo general, cuando una cuenta es bloqueada, en la mayoría de los casos se debe a que el legítimo usuario es quien fracasa en los intentos de autentificación. En lo personal, creo que el bloqueo de contraseña -aunque temporal- puede llegar a ser más peligroso de lo que se cree.
>
> **Una solución creativa**
> Tal vez, lo realmente 'molesto', es el intento de acceso ilegítimo y continuo. Los ataques por fuerza bruta si son repetidos, continuos y simultáneos, pueden también coincidir en un DDoS (nuevamente, en un escenario ideal para los atacantes). Pero esto solo sucede en ataques automatizados. Por lo tanto, se evitaría empleando algún

> método de validación 'humana'. Por ejemplo, tras 3 intentos de autentificación fallida, puedo bloquear la contraseña del usuario de forma temporal (20' por ejemplo) y a la vez, enviarle un correo al usuario, de 'desbloqueo de contraseña'. Siguiendo un enlace enviado a su correo electrónico, el usuario podría desbloquear su cuenta y seguir intentándolo (o intentar recuperar su clave).

Uso de protocolos de autentificación que no requieren contraseña

Mientras que la autentificación a través de una combinación usuario/contraseña y el uso de la autentificación de factores múltiples es generalmente considerada segura, hay casos de uso en los que no se considera la mejor opción o incluso seguro. Un ejemplo de esto son las aplicaciones de terceros que desean conectarse a la aplicación Web, ya sean desde un dispositivo móvil, algún otro sitio web, aplicaciones de escritorio u otras situaciones. Cuando esto sucede, NO es considerado seguro permitir a la aplicación de terceros almacenar la combinación de usuario/contraseña, ya que se amplía la superficie de ataque a sus manos, donde queda fuera de su control. Por esto y por otros casos de uso, hay varios protocolos de autentificación que pueden protegerlo de exponer los datos de sus usuarios a los atacantes.

OAuth

Open Authorization (OAuth) es un protocolo que permite a una aplicación autentificar a un usuario contra un servidor, sin requerir contraseñas o algún servidor externo que actúe como proveedor de identidad. Utiliza un token generado por el servidor, ofreciendo un flujo de autorización sostenido, para que un cliente tal como una aplicación móvil, pueda llamar al servidor que el usuario está utilizando el servicio.

La recomendación es usar e implementar OAuth 1.0a o OAuth 2.0, ya que a la primera versión (OAuth1.0) se la ha encontrado vulnerable a los ataques de fijación de sesión

(*session fixation*).

OAuth 2.0 se basa en HTTPS para la seguridad y actualmente es usado e implementado por las API de empresas como Facebook, Google, Twitter y Microsoft. OAuth1.0a es más difícil de usar porque requiere de bibliotecas criptográficas para las firmas digitales. Sin embargo, no se basa en HTTPS para la seguridad y, por lo tanto, puede ser más adecuado para las transacciones de mayor riesgo.

OpenId

OpenId es un protocolo basado en HTTP que utiliza proveedores de identidad para validar que un usuario es quien dice ser. Es un protocolo muy simple que permite a un proveedor de servicios de identidad un camino para el inicio de sesión único (SSO, por las siglas en inglés de "single sign-on"). Esto permite a los usuarios reutilizar una sola identidad dada a un proveedor de identidad OpenId de confianza y ser el mismo usuario en múltiples sitios web, sin la necesidad de proveer la contraseña a ningún sitio Web, exceptuando al proveedor de identidad OpenId.

Debido a su simplicidad y a que proporciona protección de contraseñas, OpenId ha sido bien aceptado. Algunos de los proveedores de identidad OpenId bien conocidos son Stack Exchange, Google, Facebook y Yahoo!

Para entornos no empresariales, OpenId es considerado seguro y frecuentemente, la mejor opción, siempre y cuando el proveedor de identidad sea de confianza.

SAML

El lenguaje de marcado para confirmaciones de seguridad (SAML, siglas en inglés de "Security Assertion Markup Language") a menudo se considera la competencia de OpenId. La versión más recomendada es la 2.0, ya que posee características muy completas y proporciona gran seguridad. Como con OpenId, SAML utiliza proveedores de identidad, pero a diferencia de éste, está basado en XML y proporciona mayor

flexibilidad. SAML está basado en redirecciones del navegador las cuales envían los datos en formato XML. A diferencia de SAML, OpenId no solo es iniciado por un proveedor de servicios, sino que también puede ser iniciado desde el proveedor de identidad. Esto permite al usuario navegar entre diferentes portales mientras que se mantienen autentificado sin tener que hacer nada, haciendo que el proceso sea transparente.

Mientras que OpenId ha tomado la mayor parte del mercado de consumo, SAML es a menudo la opción para aplicaciones empresariales. La razón de esto, frecuentemente, es que hay pocos proveedores OpenId que son considerados de clase empresarial (lo que significa que la forma en la que validan la identidad del usuario no tiene los altos estándares requeridos para la identidad de la empresa). Es más común ver SAML siendo usado dentro de la intranet de un sitio Web, a veces incluso, utilizando un servidor desde la internet como el proveedor de identidad.

En los últimos años, las aplicaciones como SAP ERP y SharePoint (SharePoint utilizando Active Directory Federation Services 2.0) deciden usar la autentificación SAML 2.0, a menudo como un método preferido para las implementaciones de inicios de sesión únicos siempre que se requiera la federación empresarial para servicios Web y aplicaciones.

Ver también: SAML Security Cheat Sheet

FIDO

La *Fast Identity Online (FIDO) Alliance* ha creado dos protocolos para facilitar la autentificación online: los protocolos *Universal Authentication Framework (UAF)* y *Universal Second Factor (U2F)*. Mientras que el protocolo UAF se enfoca en la autentificación sin contraseña, U2F permite la adición de un segundo factor de autenticación basado en contraseñas existentes. Ambos protocolos están basados en

una llave pública de modelo criptográfico desafío-respuesta.

UAF toma ventaja de las tecnologías de seguridad existentes presentes en los dispositivos de autenticación, incluyendo sensores de huellas digitales, cámaras (biométrica facil), micrófonos (biométrica de voz), Entornos de ejecución de confianza (TEE, siglas en inglés de Trusted Execution Environment), Elementos seguros (SE, siglas en inglés de Secure Elements) y otros. El protocolo está diseñado para conectar las capacidades de este dispositivo en un marco de autenticación común. UAF trabaja con ambas aplicaciones nativas y Web.

U2F aumenta la autenticación basada en contraseñas mediante un token de hardware (típicamente un USB) que almacena llaves de autenticación criptográficas y las utiliza para firmar. El usuario puede utilizar el mismo token como un segundo factor para múltiples aplicaciones. U2F trabaja con aplicaciones Web. Provee **protección contra phishing** utilizando la URL del sitio Web para buscar la llave de autentificación almacenada.

> De todos los protocolos, **oAuth es el más viable**.
> **SAML** parece estar muy pensado desde las complejas, pragmáticas y poco razonables miradas de quiénes ven en las tecnologías privativas soluciones viables. Más allá de toda opinión, es de hacer notar que **nada que sea privativo puede ser fehacientemente comprobado como seguro, ya que solo se basa en la confianza** que el programador tenga en la compañía que lo desarrolla pero no en la razón lógica. Desde mi punto de vista no parece ser una gran alternativa aunque sí funciona, lo hace desde el pragmatismo.
> **OpenId** no está nada mal, pero se basa mucho en la confianza. Es decir, se tiene que confiar en el proveedor de identidad y la confianza no es racional, sino también pragmática.
> **FIDO**, es una gran alternativa, aunque claramente costosa y solo se justifica en ámbitos demasiado puntuales (no son protocolos para emplear en el promedio de los

> sistemas).

Directrices generales para el manejo de sesiones

El manejo de sesiones está directamente relacionado a la autentificación. Las **Directrices generales para el manejo de sesiones** previamente disponibles en esta Hoja de referencias de Autentificación de OWASP han sido integradas en Session Management Cheat Sheet (NdT: hoja en proceso de traducción).

Gestión de contraseñas

Los gestores de contraseñas son programas, complementos para el navegador o servicios Web que automatizan el manejo de un gran número de diferentes credenciales, incluyendo la memorización y rellenado automático, generando contraseñas aleatorias en diferentes sitios, etc. La aplicación Web puede ayudar a los administradores de contraseñas:

- usando formularios HTML estándar para los campos de ingreso de nombres de usuario y contraseñas,
- no deshabilitando el *"copy & paste"* en los campos de los formularios HTML,
- permitiendo contraseñas muy largas,
- no usando esquemas de inicio de sesión en múltiples etapas (nombre de usuario en la primera pantalla, luego la contraseña),
- no usando esquemas de autentificación con largos scripts (JavaScript).

Recursos adicionales

Un PDF del Cheatsheet en inglés puede obtenerse aquí:

https://magic.piktochart.com/output/7003174-authentication-cheat-sheet

Autores y Editores principales

Eoin Keary

Jim Manico

Timo Goosen

Pawel Krawczyk

Sven Neuhaus

Manuel Aude Morales

Traducido al Español para OWASP, por:

Eugenia Bahit

Acerca de Eugenia Bahit

GLAMP Hacker & eXtreme Programmer especializada en **seguridad informática** e **Ingeniería Inversa de código** para el desarrollo de Software.

Docente e instructora de **programación en Python y PHP**.

Miembro de **Free Software Foundation**, **The Linux Foundation** y **OWASP**.

Creadora del *deployer* para servidores Web, **JackTheStripper** y el kernel para aplicaciones MVC modulares, **Europio Engine**.

Autora de la **Teoría sintáctico-gramatical de Objetos** y otros textos de investigación.

Fundadora de las revistas **Hackers & Developers Magazine** y **The Original Hacker**.

Sitios Web

Cursos de Programación a distancia:
www.cursosdeprogramacionadistancia.com

Web personal:
www.eugeniabahit.com

The Original Hacker:
www.originalhacker.org

Twitter:
www.twitter.com/eugeniabahit