Algoritmo HOTP (An HMAC-Based One- Time Password Algorithm)

RFC 4226

Introducción

Los password de un solo uso (OTP) son una de las formas más simples y populares de autenticación doble factor para asegurar el acceso a las redes.

HOTP se trata de un algoritmo simple para generar passwords de un solo uso, basado en HMAC (Hashed Message Authentication Code). Fue desarrollado por miembros del OATH (Open AuTHentication Initiative). Puede ser implementado por cualquier fabricante de Hardware o desarrollador de software para crear dispositivos de autenticación interoperables y agentes de software.

Requerimientos del Algoritmo

Hay algunos principios de diseño que el algoritmo debe seguir, principalmente en la usabilidad del usuario final y la capacidad para que el algoritmo sea implementado en Hardware de bajo costo.

Requerimiento 1 - el algoritmo debe ser secuencial o basado en contador. Uno de los objetivos es poder embeber el algoritmo HOTP in dispositivos como Java Smart cards, USB dongles, y GSM Sim cards.

Requerimiento 2: La implementación del algoritmo en hardware debería ser económica, minimizando requisitos de batería, cantidad de botones, poder computacional, y tamaño de la pantalla LCD.

Requerimiento 3: el algoritmo debe poder trabajar con tokens que no soportan ninguna entrada numérica, pero podría también ser usado por dispositivos más sofisticados como PIN-pads seguros.

Requerimiento 4: el valor mostrado en el token debe ser fácil de leer y fácil de ingresar por el usuario. Esto requiere que el valor HOTP tengo un tamaño razonable (ni muy pequeño ni muy grande). El valor HOTP debería tener al menos 6 dígitos. También es deseable que este valor sea sólo numérico, así sería más fácil de ingresar en dispositivos como celulares.

Requerimiento 5: deben existir mecanismos fáciles de usar para resincronizar el contador.

Requerimiento 6: El algoritmo debe usar una clave secreta fuerte., cuyo tamaño debe ser al menos 128 bits. La RFC recomienda que la longitud de la clave secreta sea de 160 bits.

Notaciones y Símbolos

Algunos símbolos utilizados por el algoritmo son:

C: es el factor de movimiento, es decir, el valor del contador (8 bytes). Este contador debe estar sincronizado entre el generador de HOTP (o sea, el cliente) y el validador de HOTP (o sea, el servidor).

K: es la clave secreta entre el cliente y el servidor. Cada generador de HOTP posee un K único, diferente al resto de los clientes.

T: parámetro de regulación: el servidor rechazará conexiones de un usuario después de “T” intentos de autenticación incorrectos.

s: parámetro de resincronización: el servidor intentará verificar un valor HOTP recibido dentro de “s” consecutivos valores del contador.

Digit: es el número de digitos de un valor HOTP, se trata de un parámetro del sistema.

Descripción General

El algoritmo HOTP utiliza dos cosas, por un lado un contador cuyo valor se va incrementando y por el otro una clave simétrica estática conocida sólo por el token y el servicio de validación. Para crear un valor HOTP, se utiliza en la mayoría de los casos el algoritmo HMAC-SHA-1 (definido en la RFC 2104).

La salida de HMAC-SHA-1 es un valor de 160 bits, el cual debe ser truncado en un valor fácil de ingresar por el usuario final. La fórmula para obtener un valor HOTP es la siguiente:

HOTP(K,C) = Truncate( HMAC-SHA-1(K,C) )

Donde Truncate representa la función que se encarga de convertir un valor HMAC-SHA-1 en un valor HOTP.

La clave (K) y el contador (C) son hasheados en un high-order-byte-first.

Los valores HOTP obtenidos por el generador HOTP son tratados como big endian.

Generando valores HOTP

Se puede describir esta operación en 3 pasos:

Paso 1: generar un valor HMAC-SHA-1.

HS = HMAC-SHA-1( K,C )

Donde HS es un String de 20 bytes.

Paso 2: Generar un String de 4 bytes (a través de Truncación Dinámica – Dynamic Truncation - DT).

Sbits = DT (HS)

Donde DT retorna un String de 31 bits.

Paso 3: computar un valor HOTP.

Snum = StringToNum(Sbits)

Esto convierte a Sbits en un número cuyo rango está en 0…2^{31}-1

Return D = Snum mod 10^Digit

D es un número en el rango de 0…10^{Digit}-1

La función de truncamiento realiza el Paso 2 y el Paso 3, es decir, el truncamiento dinámico y la operación módulo para reducir Snum. El propósito de la técnica de truncamiento de offset (desplazamiento) dinámico es extraer un código binario dinámico de 4 bytes del resultado devuelto por HMAC-SHA-1 (el cual posee un tamaño de 160 bits, o sea 20 bytes).

DT(String) // String = String[0]...String[19]

Let OffsetBits be the low-order 4 bits of String[19]

Offset = StToNum(OffsetBits) // 0 <= OffSet <= 15

Let P = String[OffSet]...String[OffSet+3]

Return the Last 31 bits of P

La razón para enmascarar el bit más significativo de P es para evitar confusiones sobre módulos de computación con signo y sin signo. Diferentes procesadores realizan estas operaciones de forma diferente, y al enmascarar el bit con signo se elimina toda ambigüedad.

La implementación debe extraer por lo menos un código de 6 dígitos, y si es posible, 7 o 8 dígitos. Esto depende del nivel de seguridad que necesitemos.

<<<ejemplo sigue>>>>

Consideraciones de Seguridad

El ataque más posible que se podría realizar contra el algoritmo HOTP es el ataque de fuerza bruta.

Asumiendo que un adversario es capaz de observar numerosos intercambios de protocolo y recoger secuencias de valores de autenticación exitosos. Este adversario, si trata de construir una función F para generar valores HOTP basados en sus observaciones, non tendrá una ventaja significativa sobre un random guess.

La conclusión lógica es simplemente que la mejor estrategia es realizar un ataque de fuerza bruta para enumerar y probar todos los valores posibles.

La seguridad del algoritmo HOTP puede definirse en base a la siguiente fórmula:

Sec = sv/10^Digit

Donde:

* Sec es la probabilidad de éxito del atacante.
* s es el tamaño de ventana de sincronización look-ahead.
* v es la cantidad de intentos de verificación.
* Digit es el número de dígitos de los valores HOTP.

Podemos jugar con s, T (el parámetro de regulación que limita el número de intentos por un atacante) y Digit hasta lograr cierto nivel de seguridad, preservando aún la usabilidad del sistema.

Requerimientos de Seguridad

Cualquier algoritmo OTP es solamente tan seguro como la aplicación y los protocolos de autenticación que la implementan. Sin embargo, se puede analizar aisladamente la seguridad de estos algoritmos. Los parámetros T y s tienen un significativo impacto en la seguridad. Es importante remarcar que el algoritmo HOTP no es un sustituto de la encriptación y no provee privacidad de transmisión de datos. Se deben utilizar otros mecanismos para defenderse de ataques destinados a corromper la confidencialidad y la privacidad de las transacciones.

1. Requerimientos del protocolo de autenticación: a continuación, se nombran algunos requerimientos para un protocolo P que implementa HOTP como método de autenticación entre un proveedor y un verificador:
   1. RP1: P debe soportar la autenticación de dos factores, es decir, la comunicación y verificación de algo que usted conoce (código secreto, como un password, pin code, etc.) y algo que usted tiene (token). El código secreto es solamente conocido por el usuario y usualmente es ingresado con el valor OTP con el propósito de autenticarse.
   2. RP2: P no debería ser vulnerable a ataques de fuerza bruta. Esto implica que el servidor debe realizar ciertas validaciones y bloqueos.
   3. RP3: P debería ser implementado a través de un canal seguro para proteger la privacidad de los usuarios y evitar ataques de repetición (fuerza bruta).
2. Validación de los valores HOTP: el cliente HOTP (token de hardware o software) incrementa su contador y luego calcula el próximo valor HOTP. Si el valor recibido por el servidor de autenticación machea con el valor calculado por el cliente, el valor HOTP es validado. En este caso, el servidor incrementa el en uno el valor del contador.

Si el valor recibido por el servidor no machea con el valor calculado por el cliente, el servidor inicia el protocolo de resincronización (look-ahead window) antes de que el mismo solicite otro password.

Si la resincronización falla, el servidor solicita otro valor de autenticación, hasta que se alcanza la cantidad máxima de intentos autorizados.

Si la cantidad máxima de intentos autorizados se supera, el servidor debería bloquear la cuenta e iniciar un procedimiento para informarle al usuario.

1. Estrangulamiento en el servidor: truncar el valor HMAC-SHA-1 a un valor más corto puede dar pie a ataques de fuerza bruta. Sin embargo, el servidor de autenticación necesita detectar y parar estos ataques.

La RFC recomienda setear un parámetro de estrangulamiento T, el cual define el número máximo de posibles intentos para la validación de OTP. El servidor gestiona contadores individuales por cada dispositivo HOTP para tomar nota de cualquier intento fallido. La RFC recomienda que T no sea demasiado grande, particularmente si el método de resincronización usado en el servidor está basado en ventanas, y el tamaño de la venta es demasiado grande. T debería setearse tan bajo como sea posible, siempre garantizando que la usabilidad no se vea afectada.

Otra opción sería implementar un esquema de delay para evitar ataques de fuerza bruta. Luego de cada intento fallido, el servidor de autenticación esperaría por un tiempo determinado, con una distribución exponencial, es decir, si en el primer intento fallido esperé 5 segundos, en el segundo consecutivo esperaré 10.

1. Resincronización del contador: Aunque el valor del contador del servidor solo se incrementa después de una autenticación HOTP exitosa, el contador en el token se incrementa cada vez que el usuario solicita un nuevo HOTP. Debido a esto, los valores de contador en el servidor y en el token pueden estar fuera de sincronización.

La RFC recomienda setear un parámetro look-ahead “s” en el servidor, el cual define el tamaño de la ventana look-ahead. En pocas palabras, el servidor puede recalcular los valores del próximo valor HOTP y verificarlos contra el cliente HOTP recibido.

La sincronización de los contadores en este escenario simplemente requiere que el servidor calcule el proxímo valor HOTP y determine si hay un macheo. Opcionalmente, el sistema debe requerir que el usuario envíe una secuencia de valores HOTP (por ejemplo: 5 valores consecutivos) con el propósito de resincronizarse, ya que forjar una secuencia de valores de HOTP consecutivos es aún más difícil que adivinar un solo valor de HOTP.

El límite superior establecido por el parámetro s garantiza que el servidor no compruebe los valores de HOTP para siempre (lo que provoca un ataque de denegación de servicio) y también restringe el espacio de posibles soluciones para un atacante que intente fabricar valores de HOTP. s DEBE establecerse lo más bajo posible, al mismo tiempo que se asegura de que la usabilidad no se vea afectada.

1. Gestión de claves secretas: Las operaciones relacionadas con claves secretas utilizados para generar y verificar valores OTP deben realizarse de forma segura, a fin de mitigar los riesgos de fugas de información sensible.

Se pueden considerar dos diferentes caminos para generar y almacenar de forma segura claves secretas in los sistemas de validación:

* Generaciones determinísticas: las claves secretas se derivan de una semilla maestra, tanto en las etapas de aprovisionamiento como de verificación, y se generan sobre la marcha siempre que sea necesario.
* Generaciones aleatorias: las claves secretas son generados aleatoriamente en la etapa de aprovisionamiento y deben almacenarse inmediatamente y mantenerse seguros durante su ciclo de vida.

Generación Determinística

Una posible estrategia es derivar las claves secretas desde una clave maestra. La clave maestra debe almacenarse únicamente en el servidor. Se DEBE usar un dispositivo a prueba de manipulaciones para almacenar la clave maestra y obtener los secretos compartidos de la llave maestra y cierta información pública. El principal beneficio sería evitar la exposición de los secretos compartidos en cualquier momento y también evitar requisitos específicos de almacenamiento, ya que los secretos compartidos podrían generarse a pedido cuando sea necesario en el momento del aprovisionamiento y la validación.

Podemos distinguir dos casos diferentes:

* Una única claves maestra MK es usada para derivar las claves secretas; cada dispositivo HOTP posee una clave secreta diferente, K\_i = SHA-1 (MK, i), donde i representa información pública que identifica de forma única al dispositivo HOTP, como un número de serie, una identificación de token, etc. Obviamente, esto está en el contexto de una aplicación o servicio: diferentes proveedores de aplicaciones o servicios tendrán diferentes secretos y configuraciones.
* Varias claves maestras MK\_i son usadas y cada dispositivos HOTP almacena un conjunto de diferentes claves derivadas, {K\_i,j = SHA-1(MK\_i,j)}, donde j representa una porción de información pública que identifica a un dispositivo. La idea sería almacenar únicamente la calve maestra activa en el servidor de validación, en el Módulo de Seguridad de Hardware (HSM), y guardar en un lugar seguro, usando métodos de compartición de claves. En este caso, si una clave maestra MK\_i se vé comprometida, es posible cambiar a otra clave sin reemplazar todos los dispositivos.

El inconveniente en el caso determinista es que la exposición del secreto maestro obviamente permitiría a un atacante reconstruir cualquier secreto compartido basado en información pública correcta. Se requerirá la revocación de todos los secretos o el cambio a un nuevo conjunto de secretos en el caso de múltiples claves maestras.

Por otro lado, el dispositivo utilizado para almacenar las claves maestras y generar los secretos compartidos DEBE ser resistente a alteraciones. Además, el HSM no estará expuesto fuera del perímetro de seguridad del sistema de validación, lo que reduce el riesgo de fugas.

Generación Aleatoria

Las claves secretas son generadas aleatoriamente. Un generador aleatorio requiere una fuente de aleatoriedad natural. Prácticamente, hay dos posibles caminos a considerar para la generación de los secretos compartidos:

* Generadores basados en Hardware: explotan la aleatoriedad que ocurre en los fenómenos físicos. Una buena implementación puede basarse en osciladores e incorporarse de tal forma que los ataques activos sean más difíciles de realizar.}
* Generadores basados en Software: diseñar un buen generador aleatorio de software no es una tarea fácil. Una implementación simple, pero eficiente, debe basarse en varias fuentes y aplicar a la secuencia muestreada una función de un solo sentido, como SHA-1.

La RFC recomienda elegir productos probados.

También recomienda almacenar las claves compartidas de manera segura, encriptándolas y sólo exponiéndolas cuando sea necesario: por ejemplo, la clave secreta es desencriptada cuando necesita verificarse un valor HOTP, y vuelta a reencriptar inmediatamente para limitar la exposición en la RAM por un período corto de tiempo. Los datos almacenados que contienen la clave secreta debe estar en un área segura, para evitar tanto como sea posible los ataques directos en los sistemas de validación y las bases de datos que las contienen.

Particularmente, el acceso a las claves secretas se debería limitar a programas y procesos requeridos por el sistema de validación únicamente.

Claves Secretas Compuestas

Sería deseable incluir incluir factores de autenticación adicionales en la clave secreta K. Estos factores adicionales pueden consistir en cualquier dato conocido en el token pero no que no sea fácilmente obtenido por otros. Por ejemplo:

* PIN o Password obtenido por el usaurio en el token.
* Número de teléfono.
* Cualquier identificador único disponible programáticamente en el token.

En este escenario, la clave secreta compuesta K es construida durante el proceso de aprovisionamiento desde una semilla el valor de una semilla aleatorio combinada con otra o más factores de autenticación adicionales. El servidor podría generar a pedido o almacenar secretos compuestos, en cualquier caso, dependiendo de la elección de implementación, el token solo almacena el valor de inicialización. Cuando el token realiza el cálculo de HOTP, calcula K a partir del valor de inicialización y los valores de entrada o de entrada local de los otros factores de autenticación.

El uso de claves secretas compuestas pueden hacer más fuerte los sistemas de autenticación basados en HOTP a través de la inclusión factores de autenticación adicionales en el token.

En la medida en que el token es un dispositivo confiable, este enfoque tiene el beneficio adicional de no requerir la exposición de los factores de autenticación (como el PIN de entrada del usuario) a otros dispositivos.

Autenticación Bidireccional

el cliente HOTP también podría utilizarse para autenticar el servidor de validación, alegando que es una entidad genuina que conoce el secreto compartido.

Como el cliente HOTP y el servidor están sincronizados y comparten el mismo secreto (o un método para recalcularlo), se podría implementar un protocolo simple de 3 pasos:

1. El usuario final ingresa el TokenID y un primer valor OTP (OTP1);
2. El servidor chequea OTP1 y si es correcto, enviar devuelta OTP2;
3. El usuario final chequea OTP2 usando su dispositivo HOTP y si es correcto, utiliza el sitio web.

Obviamente, toda la comunicación OTP debe tener lugar sobre un canal seguro, por ejemplo: SSL/TLS, Ipsec connections.