Hardening Teamwork Cloud

On this page:

Scripts

The following are the script files used in this hardening guide:

harden_cassandra_ports.sh

twc.java.security

upgrade_tomcat_webapp.sh

upgrade_jdk_webapp.sh

Introduction

The default shipping configuration of Teamwork Cloud is not a hardened configuration.

When hardening an installation, there are variables which can render the installation inoperative, such as incompatibility of the supported ciphers in a certificate and the supported ciphers in the hardened configuration.

Furthermore, the default configurations assume that the deployment is behind a secure infrastructure, and therefore required ports are globally allowed.

Since some of Teamwork Cloud's infrastructure rely on available components, newly discovered vulnerabilities need to be mitigated during the life-cycle of the installation.

Below, we will cover, for each component, the potential exploitable vulnerabilities of the different components, as well as various steps to mitigate depending on the policies of the deploying organization.

Cassandra

Port Access

When installing on Linux using our deployment scripts, all of the ports required by Cassandra for inter-node communication, as well as for the Teamwork Cloud nodes to communicate with Cassandra nodes are opened globally. This configuration is deployed mostly to facilitate testing of the environment upon installation, prior to taking any measures to harden the installation. If we check the firewall upon installation, we will see an output similar to the one below:

```
# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0
  sources:
  services: cassandra lmadmin ssh twcloud
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

In our deployment we create a firewall service definition to facilitate management of the rules. This file is located in /etc/firewalld/services/cassandra.xml, and contains the following

```
# cat /etc/firewalld/services/cassandra.xml
<?xml version="1.0" encoding="utf-8"?>
<service version="1.0">
    <short>cassandra</short>
    <description>cassandra</description>
    <port port="7000" protocol="tcp"/>
    <port port="7001" protocol="tcp"/>
        <port port="9042" protocol="tcp"/>
        <port port="9160" protocol="tcp"/>
        <port port="9142" protocol="tcp"/>
</service>
```

The first step in securing Cassandra is to limit traffic only to Cassandra and Teamwork Cloud Nodes. In the example below, we have a single node Cassandra/Teamwork Cloud installation, with its IP address set to 10.254.254.56

As can be seen below, the firewall configuration has been modified to only allow access from itself.

```
# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0
  sources:
  services: lmadmin ssh twcloud
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
        rule family="ipv4" source address="10.254.254.56" service name="cassandra" accept
```

The process to follow is to remove the general firewall allowance to the cassandra service. After that, we want to ensure that if direct port rule assignments were made, they are removed. Finally, we want to create a set of rich rules which will allow access only to the required nodes.

If your deployment consists of a single-node or multi-node cluster where both Teamwork Cloud and Cassandra reside on the same nodes, we can automate this process by using the following script.

harden_cassandra_ports.sh

```
#!/bin/bash
echo "Hardening Cassandra Ports"
echo "Creating Cassandra firewall service profile"
cat <<EOF |  tee /etc/firewalld/services/cassandra.xml  &> /dev/null
<?xml version="1.0" encoding="utf-8"?>
<service version="1.0">
    <short>cassandra</short>
    <description>cassandra</description>
    <port port="7000" protocol="tcp"/>
    <port port="7001" protocol="tcp"/>
        <port port="9042" protocol="tcp"/>
        <port port="9160" protocol="tcp"/>
        <port port="9142" protocol="tcp"/>
</service>
EOF
echo "Removing existing firewall rules on the cassandra ports or service profile"
sleep 5
zone=$(firewall-cmd --get-default) &> /dev/null
firewall-cmd --zone=$zone --remove-port=7000/tcp --permanent  &> /dev/null
firewall-cmd --zone=$zone --remove-port=7001/tcp --permanent  &> /dev/null
firewall-cmd --zone=$zone --remove-port=7199/tcp --permanent  &> /dev/null
firewall-cmd --zone=$zone --remove-port=9042/tcp --permanent  &> /dev/null
firewall-cmd --zone=$zone --remove-port=9160/tcp --permanent  &> /dev/null
firewall-cmd --zone=$zone --remove-port=9142/tcp --permanent  &> /dev/null
firewall-cmd --zone=$zone --remove-service=cassandra --permanent  &> /dev/null
echo "Creating ruch rules for Cassandra nodes discovered via nodetool gossipinfo"
set -f
local_list=($(nodetool gossipinfo | grep '/' | cut -f 2 -d /))
set +f
for i in "${local_list[@]}" ; do
        cmd=" firewall-cmd --zone=$zone --add-rich-rule='rule family=\"ipv4\" source address=\"$i\" service
name=\"cassandra\" accept' --permanent  &> /dev/null"
        echo $cmd
        eval $cmd
done
firewall-cmd --reload  &> /dev/null
```

The above script is structured in such a way that it will output the rich rule commands to the screen to facilitate copying and modifying to add nodes, such as would be the case if you have Teamwork Cloud nodes which are not part of the Cassandra cluster.

The above script should be executed on all nodes of a multi-node Cassandra cluster, with all nodes being in an operational state.

Windows Users: Create firewall rules restricting access on the aforementioned ports only to authorized nodes - Cassandra nodes and Teamwork Cloud Nodes.

Replication Strategy

Teamwork Cloud presently does not support Multi-DC replication strategies. As such, the environment must be configured with SimpleStrategy.

Cassandra Authentication

By default, Cassandra is deployed with the AllowAllAuthenticator. This authenticator, as the name implies, is an anonymous authenticator which performs no checks.

If you require authenticated connections, you will need to make changes to your cassandra.yaml and change the authenticator to PasswordAuthenticator. If you are running a multi-node Cassandra cluster, you need to change the replication factor of the system_auth keyspace.

Detailed instructions can be found at https://docs.datastax.com/en/ddacsecurity/doc/ddacsecurity/secureConfigNativeAuth.html

After making the changes to cassandra, you will need to update the Teamwork Cloud configurations to utilize authentication

The required changes are as follows:

application.conf

```
                         # Enable this section to enable the cassandra authentication.
                         authentication-enabled = true
                         username = newcassandrauser
                         password = newcassandrapassword
```

authserver.properties

```
cassandra.username=newcassandrauser
cassandra.password=newcassandrapassword
```

where newcassandrauser and newcassandrapassword correspond to the user and credentials which you configured in Cassandra

Cassandra Encrypted Connections

Teamwork Cloud presently does not support encrypted communications to Cassandra

Data Encryption at Rest

Open source Cassandra does not support encryption at rest. If you require encryption at rest, you need to use a disk encryption layer.

Teamwork Cloud

Protocols and Ciphers

Teamwork Cloud consists of 3 Java based services - Teamwork Cloud (twcloud), Authserver (authserver) and WebApp (webapp).

twcloud and authserver require Java 8 (its location varies depending on how it was deployed), whereas webapp uses a bundled Java 12, located in <installation_directory>/WebAppPlatform/jre/.

Therefore, in order to harden these services we must begin by hardening the JVM. The default settings for the JVM are located in java.security.

We can check the ciphers/protocols being used by the applications using nmap (version 7.x) or TestSSLServer.jar, available from https://community.rsa.com/docs/DOC-45511

As an example, below is a scan using both tools against a default installation. In this example, we will be testing port 8111, the twcloud port.

```
# nmap --script ssl-enum-ciphers -p 8111 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2020-04-21 17:32 MDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00014s latency).

PORT     STATE SERVICE
8111/tcp open  unknown
| ssl-enum-ciphers:
|   TLSv1.0:
|     ciphers:
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: client
|     warnings:
|       Key exchange (dh 1024) of lower strength than certificate key
|   TLSv1.1:
|     ciphers:
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: client
|     warnings:
|       Key exchange (dh 1024) of lower strength than certificate key
|   TLSv1.2:
|     ciphers:
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (dh 1024) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secp256r1) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A
|       TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: client
|     warnings:
|       Key exchange (dh 1024) of lower strength than certificate key
|_  least strength: A

Nmap done: 1 IP address (1 host up) scanned in 0.76 seconds

# java -jar TestSSLServer.jar 127.0.0.1 8111
Supported versions: TLSv1.0 TLSv1.1 TLSv1.2
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
  TLSv1.0
     RSA_WITH_AES_128_CBC_SHA
     DHE_RSA_WITH_AES_128_CBC_SHA
     TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
  (TLSv1.1: idem)
  TLSv1.2
     RSA_WITH_AES_128_CBC_SHA
     DHE_RSA_WITH_AES_128_CBC_SHA
     RSA_WITH_AES_128_CBC_SHA256
     DHE_RSA_WITH_AES_128_CBC_SHA256
     TLS_RSA_WITH_AES_128_GCM_SHA256
     TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
     TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
     TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
     TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
----------------------
Server certificate(s):
  a241e1c3b957bd14e6e0242fd012f75853eef243: CN=X.X.X.X
----------------------
Minimal encryption strength:     strong encryption (96-bit or more)
Achievable encryption strength:  strong encryption (96-bit or more)
BEAST status: vulnerable
CRIME status: protected
```

As can be observed above, the default configuration using OpenJDK 1.8.0_242 is allowing TLS v1.0 and v1.1, which are deprecated. Additionally, we can see that several key exchanges are taking place using dh1024.

We then proceed to harden the configuration.

Since we are dealing with ciphers, you need to make sure that you do not disable a cipher required by your certificates.

After hardening the VM, we end up with a different set of allowed ciphers and protocols, as shown below.

```
# nmap --script ssl-enum-ciphers -p 8111 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-21 17:44 MDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00015s latency).

PORT     STATE SERVICE
8111/tcp open  unknown
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secp256r1) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A
|       TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: client
|_  least strength: A

Nmap done: 1 IP address (1 host up) scanned in 0.95 seconds

# java -jar TestSSLServer.jar 127.0.0.1 8111
Supported versions: TLSv1.2
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
  TLSv1.2
     RSA_WITH_AES_128_CBC_SHA256
     TLS_RSA_WITH_AES_128_GCM_SHA256
----------------------
Server certificate(s):
  71ed3969e41a94877c51aacb87d995af4a12b6d9: CN=x.x.x.x
----------------------
Minimal encryption strength:     strong encryption (96-bit or more)
Achievable encryption strength:  strong encryption (96-bit or more)
BEAST status: protected
CRIME status: protected
```

The process of hardening the JVM requires making some changes to the java.security file. While these can be made directly, the downside is that if you upgrade your JVM, you will have to reapply your changes.

However, we can place our modifications in our own file, and simply pass a parameter to the JVM upon invocation so that it will apply our changes.

For example, we can create a file */home/twcloud/twc.java.security*, and pass a parameter to the JVM in the form of *-Djava.security.properties=/home/twcloud/twc.java.security*

Our hardened security settings are as shown below:

twc.java.secuirty

```
jdk.tls.disabledAlgorithms=SSLv3, TLSv1, TLSv1.1, RC4, DES, MD5withRSA, DH keySize < 2048, \
    EC keySize < 224, 3DES_EDE_CBC, anon, RSA keySize < 2048, SHA1, DHE, NULL
jdk.tls.ephemeralDHKeySize=2048
jdk.tls.rejectClientInitiatedRenegotiation=true
```

To apply these settings we need to make changes in 3 locations

For the Teamwork Cloud service, under Linux, you need to edit <install_directory>/twcloud.ini and add a line as shown below:

```
.
.
-Dorg.jboss.netty.epollBugWorkaround=true
-Dio.netty.epollBugWorkaround=true
-Djava.security.properties=/home/twcloud/twc.java.security
```

On Windows, you need to edit the registry key Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Apache Software Foundation\Procrun 2.0\TeamworkCloud\Parameters\Java\Options and append the setting pointing to your security overrides to the bottom of the settings.

For the Authserver service, you need to edit <install_directory>/AuthServer/authserver-run by inserting the directive before the call to the authentication-server-XXXXXX.jar as shown below:

```
$JAVA -jar \
-Duser.home.dir=$TWCLOUD_OWNR_HOME \
-Dfile.encoding=utf-8 \
-Dauthentication-config=./config \
-Dspring.config.location=./config/authserver.properties \
-Djava.security.properties=/home/twcloud/twc.java.security \
authentication-server-XXXXXX.jar "$@"
```

On Windows, you need to edit the registry key Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Apache Software Foundation\Procrun 2.0\AuthServer\Parameters\Java\Options and append the setting pointing to your security overrides to the bottom of the settings.

For the Webapp service, under Linux you need to edit <install_directory>/WebAppPlatform/bin/setenv.sh and add the directive to the JVM_OPTS variable as shown below:

```
JVM_OPTS="-server -XX:+UseParallelGC  -Xms4096M -Xmx8192M -Djava.security.properties=/home/twcloud/twc.java.
security"
```

On Windows, you need to edit the registry key Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Apache Software Foundation\Procrun 2.0\WebApp\Parameters\Java\Options and append the setting pointing to your security overrides to the bottom of the settings.

JMX

By default, the twcloud service activates a JMX remote port to facilitate application monitoring. The default configuration does not contain any form of authentication.

On Linux, the configuration is located in <install_dir>/twcloud.ini.

On Windows, it is located in registry key Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Apache Software Foundation\Procrun 2.0\TeamworkCloud\Parameters\Java\Options.

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=2468
-Dcom.sun.management.jmxremote.rmi.port=2468
-Dcom.sun.management.jmxremote.local.only=false
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

These settings can be removed, thereby removing JMX remote access.

If you would like to allow remote JMX access, but require authentication, you can do so by adding settings. For complete documentation, please refer to the Java documentation.

As an example, the below configuration adds password authentication

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=2468
-Dcom.sun.management.jmxremote.rmi.port=2468
-Dcom.sun.management.jmxremote.local.only=false
-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.password.file=/home/twcloud/jmx.password
-Dcom.sun.management.jmxremote.access.file=/home/twcloud/jmx.access
-Dcom.sun.management.jmxremote.ssl=false
```

As can be seen, we are pointing to a set of files (/home/twcloud/jmx.password and /home/twcloud/jmx.access) which control who can access these files.

The vulnerability vector is one whereby JMX could be exploited to execute code. To prevent this, we allow only an authenticated user (jmx.password) who has read only rights (jmx.access)

jmx.password

```
monitoring  DqzbksT4ET
```

jmx.access

```
monitoring readonly
```

In this example, we created a user (monitoring) with a password (DqzbksT4ET), who can only read values via Remote JMX, but cannot write or execute anything via JMX.

The password and access files have a very stringent ownership requirement. They need to be owned by the user running the process, and be accessible exclusively to that user.

For example, in our default installation, the twcloud user is running the twcloud service. Therefore, the files need to be owned by twcloud, and have full rights (rwx) by twcloud, and only twcloud.

```
# ll jmx.*
-rwx------. 1 twcloud twcloud 20 Mar 16 15:11 jmx.access
-rwx------. 1 twcloud twcloud 26 Apr 21 10:41 jmx.password
```

Apache Tomcat

The Webapp Platform (webapp), as deployed by our installer, runs on a bundled Apache Tomcat. As such, best practices for hardening Apache Tomcat should be followed.

Although we have already constrained ciphers and protocols at the JVM level, it is best practice to do so at the Tomcat configuration level. We also need to address issues such as secure cookies, disable XSS on foreign sites, and also remove default directories published as part of the default installation. The official Tomcat documentation covers a large portion of this (recommended reading) - https://tomcat.apache.org/tomcat-9.0-doc/security-howto.html. Additionally, there are a plethora of documents online covering all aspects of securing Tomcat to OWASP standards.

server.xml

There are various changes which can be made to *<install_dir>/WebAppPlatform/conf/server.xml* in order to harden the system.

The first step (do not do this if running on Windows) is to disable the shutdown port.

For this, you need to change

```
<Server port="8005" shutdown="SHUTDOWN">
```

to

```
<Server port="-1" shutdown="SHUTDOWN">
```

The next step is to disable the AJP connector , unless you specifically intend to use it.

For this, you need to change

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

to

```
   <!-- Define an AJP 1.3 Connector on port 8009 -->
   <!--
   <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
   -->
```

The next step is to disable the redirection on port 8080.

For this, you need to change

```
     <Connector executor="tomcatThreadPool"
                port="8080" protocol="HTTP/1.1"
                connectionTimeout="20000"
                redirectPort="8443" />
```

to

```
    <!--
     <Connector executor="tomcatThreadPool"
                port="8080" protocol="HTTP/1.1"
                connectionTimeout="20000"
                Server=" "
                redirectPort="8443" />
-->
```

Next, we will explicitly allow protocols and ciphers.

For this, you need to change

```
     <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
                sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"
                maxThreads="150" SSLEnabled="true">
       <SSLHostConfig>
         <Certificate certificateKeystoreFile="../configuration/keystore.p12"
                      certificateKeystorePassword="nomagic"
                      type="RSA" />
       </SSLHostConfig>
     </Connector>
```

to

```
      <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
                sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"
                Server=" "
                maxThreads="150" SSLEnabled="true">
       <SSLHostConfig sslProtocol="TLS"
                  protocols="TLSv1.2"
                  honorCipherOrder="true"
                  certificateVerification="none"
                  ciphers="TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
                           TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
                           TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384,
                           TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,
                           TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
                           TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
                           TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256,
                           TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,
                           TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
                           TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
                           TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384,
                           TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384,
                           TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
                           TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
                           TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256,
                           TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256" >
         <Certificate certificateKeystoreFile="../configuration/keystore.p12"
                      certificateKeystorePassword="nomagic"
                      type="RSA" />
       </SSLHostConfig>
     </Connector>
```

In the code presented above, we are showing the default configuration with the self-signed certificates. Your production configuration may be using a different certificate, so this section will differ.

Finally, we want to prevent our instance from advertising what server is being used in the event that an error is encountered.

For this, you need to go to the very bottom of the file and add the following, right above the closing </Host> tag.

```
        <!-- Suppress server name on internal error pages -->
        <Valve className="org.apache.catalina.valves.ErrorReportValve" showReport="false" showServerInfo="
false" />
      </Host>
```

web.xml

Having completed the configuration of server.xml, we will now proceed to configure *<install_dir>/WebAppPlatform/conf/web.xml.*

First, we need to enforce HSTS and prevent click-jacking.

```
    <filter>
        <filter-name>httpHeaderSecurity</filter-name>
        <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
        <async-supported>true</async-supported>
    <init-param>
      <param-name>hstsEnabled</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>hstsMaxAgeSeconds</param-name>
      <param-value>31536000</param-value>
    </init-param>
    <init-param>
      <param-name>antiClickJackingEnabled</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>antiClickJackingOption</param-name>
      <param-value>SAMEORIGIN</param-value>
    </init-param>
    </filter>


    <filter-mapping>
        <filter-name>httpHeaderSecurity</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
    </filter-mapping>
```

Next, we need to ensure that cookies are constrained to https.

```
  <!-- ==================== Default Session Configuration ================= -->
  <!-- You can set the default session timeout (in minutes) for all newly   -->
  <!-- created sessions by modifying the value below.                       -->

    <session-config>
        <session-timeout>30</session-timeout>
                <cookie-config>
                <http-only>true</http-only>
                <secure>true</secure>
                </cookie-config>
    </session-config>
```

Tomcat Installation

Having made the necessary changes to the configuration files, we will now proceed to remove all of the default applications in the tomcat distribution, which could expose our installation to external vulnerabilities.

If we look at a directory of <install_directory>/WebAppPlatform/webapps, we will see the following

```
drwxrwxr-x. 14 twcloud twcloud     4096 Apr 15 14:39 docs
drwxrwxr-x.  6 twcloud twcloud       83 Apr 15 14:39 examples
drwxrwxr-x.  5 twcloud twcloud       87 Apr 15 14:39 host-manager
drwxrwxr-x.  5 twcloud twcloud      103 Apr 15 14:39 manager
drwxrwxr-x.  3 twcloud twcloud      283 Apr 15 14:39 ROOT
drwxr-x---.  8 twcloud twcloud      117 Apr 15 14:47 webapp
-rwxrwxr-x.  1 twcloud twcloud 67742880 Oct 31 17:56 webapp.war
```

As you can see, in addition to webapp.war and and the webapp directory, there are additional directories, containing applications, which could potentially be exploited.

You want to remove docs, examples. host-manager, manager, and ROOT.

When you remove the ROOT application directory, accessing https://ip_address:8443 will no longer display the Apache Tomcat default landing page.

Upgrading Tomcat

Our installers deploy with a given version of Apache Tomcat. As vulnerabilities are exposed in Tomcat, you may be required by your organization to upgrade to a specific version.

The "code" of tomcat is the compilation of the jar files residing in <instal_dir>/WebAppPlatform/bin and <instal_dir>/WebAppPlatform/lib.

In order to "slip-stream" an upgrade without having to fully replace the Tomcat installation, you can replace the existing *.jar files in these directories with the ones from the new one.

Before doing this, you will want to make copies of these directories so you can easily revert back in case of an incompatibility with the new version.

Under Linux, assuming that you have access to the internet from the server, you can use the script below to automatically upgrade your instance to the target version.

upgrade_tomcat_webapp.sh

```
#!/bin/bash
###########################################################
#  Upgrade Tomcat Version used by WebApp Platform
#  Benjamin Krajmalnik (benjamin.krajmalnik@3ds.com)
# ################################
# This script utilizes rsync, so we will install it via yum
# Please set the tomcat version you would like to use

TOMCAT_VERSION=9.0.35

#####################################
#
#  DO NOT MODIFY ANYTHING BEYOND THIS POINT
#
#####################################

WEBAPP_ROOT=$(cat /etc/systemd/system/webapp.service | grep CATALINA_HOME_WEBAPP | cut -f 3 -d '=')
WEBAPP_OWNER=$(stat -c "%U:%G" $WEBAPP_ROOT)

#####################################
#  Install rsync
yum install rsync -y -q
####################################
# Setting up script variables
MAJOR_VERSION=$(echo $TOMCAT_VERSION | cut -d . -f 1)
TOMCAT_DOWNLOAD=http://apache.mirrors.pair.com/tomcat/tomcat-$MAJOR_VERSION/v$TOMCAT_VERSION/bin/apache-
tomcat-$TOMCAT_VERSION.tar.gz
TOMCAT_TAR=$(basename $TOMCAT_DOWNLOAD)
TOMCAT_DIR=$(basename $TOMCAT_TAR .tar.gz)

#####################################
# Begin deployment
mkdir _tmp
cd _tmp/
wget $TOMCAT_DOWNLOAD
tar -xf $TOMCAT_TAR
cd $TOMCAT_DIR
rsync -av bin/*.jar $WEBAPP_ROOT/bin/
rsync -av lib/*.jar $WEBAPP_ROOT/lib/
####################################
# Ensure proper ownership of files
chown -R $WEBAPP_OWNER $WEBAPP_ROOT
####################################
# Remove downloaded files
cd ../..
rm -fr _tmp
```

The script provided above may stop working if the Apache Tomcat distribution changes the methodology used in storing the tarfiles.

Upgrading Webapp bundled JDK

The default installation comes bundled with AdoptOpenJDK (build 12+33).

Webapp can run with Java 14.

If you wish to use it instead of the bundled version, it is located in <installation_dir>/WebAppPlatform/jre

upgrade_jdk_webapp.sh

```
#!/bin/bash
##########################################################
#  Upgrade JDK in NoMagic Webapp Platform to a newer OpenJDK
#  Benjamin Krajmalnik (benjamin.krajmalnik@3ds.com)
##########################################################

###  JRE_DOWNLOAD contains the download URL to the target OpenJDK tar archive
###  The example below upgrades the JDK to OpenJDK 14.0.1

JRE_DOWNLOAD=https://download.java.net/java/GA/jdk14.0.1/664493ef4a6946b186ff29eb326336a2/7/GPL/openjdk-14.0.1
_linux-x64_bin.tar.gz

#####  Do not modify below this point  ######
yum install wget -y -q
JRE_HOME=$(cat /etc/systemd/system/webapp.service | grep JRE_HOME | cut -f 3 -d '=')
JRE_OWNER=$(stat -c "%U:%G" $JRE_HOME)
JRE_TAR=$(basename $JRE_DOWNLOAD)
mkdir _tmp
cd _tmp
##   Download OpenJDK 14.0.1
wget $JRE_DOWNLOAD

##  Remove current JRE_HOME
rm -fr $JRE_HOME

##  Extract OpenJDK
mkdir -p $JRE_HOME
tar -xf $JRE_TAR -C $JRE_HOME --strip-components=1
chown -R $JRE_OWNER $JRE_HOME

cd ..
rm -fr _tmp
```