ITIS 6200/8200 Principles of Information Security and Privacy

FALL 2016

SEMESTER PROJECT FINAL SUBMISSION

December 11, 2016

Project Topic: Honeypots

Group Members	
Name	UNCC ID#
Jonathan Frazier	800247676
Tanya Peddi	800968024
Krutika Patil	800965896
Uma VadiveLakshmanan	800980769

Motivation Behind Tomcat Honeypot Tool

Apache Tomcat is a widely-used Java application server. From the Apache.org wiki, some large well-known companies that are known to use Tomcat server are CardinalHealth, Walmart.com, The Weather Channel, and E*Trade. This list just some of the companies that are using Tomcat in production. Our motivation behind creating a Tomcat Honeypot was twofold: first, given its widespread use, we felt there was a need for a tool such as this and second, for us as a group to gain experience with a new piece of software as only a few of us had any experience with Apache Tomcat.

Since Tomcat is a widely-used Java applet server, we felt there would be a need among organizations that use Apache Tomcat to use our honeypot to help protect the organization's production Tomcat servers. Since Tomcat is typically deployed on a host system, this deployment opens an additional avenue of attack into an organization's network. Thus, an attacker, when looking to compromise a system on an organization's network, may view it as easier to compromise the Tomcat server running on the host system to gain control of the system verses directly attacking the system itself. This is where our honeypot would come into play for an organization. By deploying our honeypot, an organization would be providing a decoy for attackers to attack which would in turn provide an organization with valuable intel that could then be used to protect the organization's real Tomcat servers.

The second reason we chose this project was to gain experience with software that as a group we were not that familiar with. Several of us in the group have used Tomcat in the past for different classes, but none of us were experts in its use or management. Thus, this project provided us with a great opportunity to learn more about a very popular server application in addition to learning concepts directly related to building a honeypot.

Technical Contributions/Experiments/Learning Experience

In general, our honeypot is like a HTTP server with a file upload capability. In designing and coding our honeypot, the Java 8.0 Javadoc proved to be invaluable to understanding the different built in Java classes which we used in the design of our honeypot. In addition, stackoverflow.com also proved to be a great resource for getting a more in-depth explanation of what the methods within the built-in Java classes do and how to properly utilize them. The inspiration to create a low interaction honeypot came from our research into Honeyd and Glastopf, which are two open source honey pots utilized today. Honeyd and Glastopf both are designed to mimic the software or applications that they are based on, and that served as our inspiration behind the Tomcat Honeypot. A low interaction honeypot that mimics the functionality of the software it is based while not actually implementing or using that software in the honeypot is not as vulnerable as the software making it easier and more secure for an organizations security team to deploy. That is what we wanted to accomplish with our Tomcat Honeypot, produce a honeypot that acted like Apache Tomcat but which would not be vulnerable to any current or future Tomcat vulnerabilities. This is important since in deploying

our honeypot, an organization is expecting it to be attacked and probed, and by only emulating Apache Tomcat services, our honeypot is easier for an organization to setup and deploy.

In coding our honeypot, it was very much a trial and error process. Most of the early experiments involved testing that our honeypot could be contacted from other hosts. Our application uses the Java class ServerSocket to listen for network connections. In preparing this portion of the project, the Java docs and tutorials on Oracle's website were very useful in explaining how to use the ServerSocket class. The next experiments we conducted was how to receive and process an HTTP request. After discovering that the Socket class provided a means to read these requests to a String object, our experiments boiled down to testing different String parsing algorithms to properly get the HTTP headers in the packet from the client and then process the clients request. Once we could read HTTP requests from the client and parse the request, we then had to experiment with creating the HTTP header for our response back to the client as well as the body of the HTTP packet which would contain the simulated Tomcat response from our honeypot. Once we learned the proper format of an HTTP response, we used a String object to store our response and use the methods in the Java Socket class to send our response to the client. Since interacting with Apache Tomcat over the network is done via HTTP, and HTTP is a standard application layer protocol, this made building our honeypot application much easier, as we only need to deal with constructing the proper HTTP response to an HTTP request.

After presenting the prototype of our honeypot, we added an authentication feature, multi-threading to handle multiple clients, the ability to upload a wider range of files, with each upload being saved in a uniquely identifiable file, and a blacklist feature to block IP address that upload files to our honeypot from continuing to interact with the honeypot. The implementing of the multi-threading feature and the ability to upload a wider range of files proved simple to get up and running compared to the authentication and blacklist feature. The authentication features required lots of trial and error since to authenticate we had to have our honeypot application send the right sequence of HTTP request with the proper HTTP status codes to tell the client that a username and password would be required to access our honeypot. Our honeypot then had to be coded to parse from the client's HTTP request the username and password field and check that against the hard-coded username and password in our honeypot. To get the blacklist feature working we had to experiment with having our honeypot check a log file of blacklisted IP addresses that the honeypot creates when the honeypot receives each new network connection. The honeypot then checks the IP address of the connection with the IP addresses in the blacklist log file, if the new connection's IP address is found in the blacklist log, the connection is denied, otherwise the honeypot will move on to process the HTTP request received on the new connection.

In completing our Tomcat Honeypot project we learned many things. In preparing to design our project, we learned about the management of Apache Tomcat over a network connections, and the different options and features that Apache Tomcat allowed a user to configure on the Tomcat server via a network connection. In coding our honeypot, we learned about how to build an HTTP server using Java, how to construct and format HTTP requests and responses, and

how to parse HTTP request in Java. Since our honeypot only mimicked that command lines commands for Apache Tomcat, we became familiar with the Linux command line utilities wget and cURL, and how to use these utilities to create HTTP requests. Additionally, since our project was on honeypots, we obtained a good understanding of what a honeypot is, the typical features that a honeypot implements, as well as some of the latest honeypot applications in use today. And since we used Java as our programming language, we were able to further improve and hone our coding skills in Java.

Apache Tomcat Honeypot

We have designed a honeypot that emulates some basic services of Apache Tomcat server. Our honeypot will be categorized as a low interaction production honeypot. As a low interaction honeypot, our tool will provide only limited services for the attacker to interact with. Our honeypot's interaction with the attacker will be achieved utilizing scripts that emulate interaction with Tomcat server instead of an actual installation of Apache Tomcat. The honeypot will be able to gather information on the attacker, such as time of connection to the honeypot, IP address and port number where the attack originates from, port number that the attacker connects to on our honeypot, and some limited functionality to allow the attacker to upload a file to our "server" that the attacker plans to use to gain control of the system.

Prerequisites:

1. Java: NetBeans IDE.
2. Nmap: It is the security scanner which the attacker uses to scan and find the open ports and services in the target machine. Nmap can be downloaded in the following link, https://nmap.org/download.html
3. Wget , cURL – Command line tools for getting and sending files and commands. Attacker uses these tools to send the manager commands(Tomcat) to the target machine.

   Examples for wget and cURL commands:

   Wget http://tomcat:tomcat@<IPADDRESSTOMCAT>:8080/manager/text/serverinfo

   curl -T test.txt --user tomcat:tomcat -X PUT
   http://localhost:8080/manager/text/deploy?path=/test

Test Cases:

Note: Before trying the test cases, please read the “Linux Command Line Utilities Explained” section after this section to understand the commands we show in our test cases as well as the parameters we use for those commands.

Test Case Description:

Test Case ID	**Test Case Description**	**Requirements for the executing the test cases.**
TC1	Honeypot receives the network connection on port 8080 and connection gets recorded in a log file created by the honeypot.	1. It is expected that the attacker knows the IP address of the host machine in which honey pot is installed. 2. Attacker machine with Nmap installed. 3. Host machine with honeypot installed 4. It is expected for the tester to know the path in which the application is installed to locate the log file.
TC2	Honeypot interacts with the attacker by responding to basic commands, providing correct responses for services emulated by the honey pot.	1. Attacker machine with wget installed 2. Host machine with honeypot installed.
TC3	Honeypot accepts the file from the attacker and securely stores the file on the host system of the project	1. A test file needs to be created to be sent from attacker's machine to tomcat server (honeypot application). 2. Host machine with honeypot installed. 3. Attacker machine with cURL installed

Test Case 1:

Test case ID	Test Step ID	Test Data	Test Step Description	Expected Result
TC1	TS1		Open the command prompt in the attacker system	Command prompt opens up
TC1	TS2	nmap -sS -sV <IPADDRESSofTOMCATSYSTEM>	Enter the Nmap command , as mentioned in the test data and hit enter	Nmap is expected to start scanning the host system and display all the open ports , their state and services.
TC1	TS3		Close the command prompt	Command prompt closes.
TC1	TS4		Open the folder in which honeypot is located.	Log file that containing the timestamp , Local IP Address,Port Number , Attacker's IP Address and port number is expected to be saved in the folder.

Test Case 2:

Test Case ID	Test Step ID	Test Data	Test Step Description	Expected Result
TC2	TS1		Open the command prompt in the attacker system	Command prompt opens up
TC2	TS2	Test Data for TC2	Enter any of the tomcat manager commands available in the document and hit enter	Response(as available in the document) to the entered command is expected to be displayed in the command prompt
TC2	TS3		Repeat step TS2 for all the server details that the attacker needs from the tomcat server(Honey pot)	Response(as available in the document) to the entered command is expected to be displayed in the command prompt
TC2	TS4		Close the command prompt	Command prompt closes.

Test data for TC 2:

Sno	Command Usage	Command	Response
1	List Currently Deployed Applications	wget http://tomcat:tomcat@localhost:8080/manager/text/list -O - -q	OK - Listed applications for virtual host localhost /webdav:running:0:webdav /examples:running:0:examples /manager:running:0:manager /:running:0:ROOT /test:running:0:test##2 /test:running:0:test##1
2	Reload An Existing Application	wget http://tomcat:tomcat@localhost:8080/manager/text/reload?path=/examples -O - -q	OK - Reloaded application at context path /examples
3	List OS And JVM Properties	wget http://tomcat:tomcat@localhost:8080/manager/text/serverinfo -O - -q	OK - Server info Tomcat Version: Apache Tomcat/8.0.38\n" OS Name: Linux\n" OS Version: 4.8.4-200.fc24.x86_64\n" OS Architecture: amd64\n" VM Version: 1.8.0_111-b16\n" JVM Vendor: Oracle Corporation\n";
4	List Available Global JNDI Resources	wget http://tomcat:tomcat@localhost:8080/manager/text/resources -O - -q	OK - Listed global resources of all types
5	Session Statistics	wget http://tomcat:tomcat@localhost:8080/manager/text/sessions?path=/examples -O - -q	OK - Session information for application at context path /examples Default maximum session inactive interval 30 minutes
6	Expire Sessions	wget http://tomcat:tomcat@localhost:8080/manager/text/expire?path=/examples&idle=10 -O - -q	OK - Session information for application at context path /examples Default maximum session inactive interval 30 minutes >10 minutes: 0 sessions were expired
7	Start an Existing Aplication	wget http://tomcat:tomcat@localhost:8080/manager/text/start?path=/examples -O - -q	OK - Started application at context path /examples

8	Stop an Existing Application	wget http://tomcat:tomcat@localhost:8080/manager/text/stop?path=/examples -O - -q	OK - Stopped application at context path /examples
9	Undeploy an Existing Application	wget http://tomcat:tomcat@localhost:8080/manager/text/undeploy?path=/examples -O - -q	OK - Undeployed application at context path /examples
10	Connector SSL/TLS Diagnostics	wget http://tomcat:tomcat@localhost:8080/manager/text/sslConnectorCiphers -O - -q	OK - Connector / SSL Cipher information Connector[HTTP/1.1-8080] SSL is not enabled for this connector Connector[HTTP/1.1-8443]
11	Save Configuration	wget http://tomcat:tomcat@localhost:8080/manager/text/save -O - -q	FAIL - No StoreConfig MBean registered at [Catalina:type=StoreConfig]. Registration is typically performed by the StoreConfigLifecycleListener.
12	Deploy A New Application Archive(WAR) Remotely	curl -T test.txt --user tomcat:tomcat -X PUT http://localhost:8080/manager/text/deploy?path=/test	OK - Deployed application at context path /foo
13	Deploy A Previuosly Deployed WebApp	curl -T test.txt --user tomcat:tomcat -X PUT http://localhost:8080/manager/text/deploy?path=/test&tag=footag	OK - Deployed application at context path /foo
14	Deploy A WAR by URL	curl -T test.txt --user tomcat:tomcat -X PUT http://localhost:8080/manager/text/deploy?path=/test&war=file:/path/to/foo	OK - Deployed application at context path /foo
15	Deploy a WAR from Host appBase	curl -T test.txt --user tomcat:tomcat -X PUT http://localhost:8080/manager/text/deploy?war=test	OK - Deployed application at context path /foo

Test Case 3:

Test Case ID	Test Step ID	Test Data	Test Step Description	Expected Result
TC3	TS1		Open the command prompt in the attacker system	command prompt opens up
TC3	TS2	curl -T test.txt --user tomcat:tomcat -X PUT http://localhost:8080/manager/text/deploy?path=/test	Enter the Application deploy command " " , and hit enter	Application deployed message is expected to be displayed in the command prompt
TC3	TS3		Close the command prompt	Command prompt closes.
TC3	TS4		Open the folder in which honeypot is located.	The file received from the attacker is expected to be saved in the format source-Downloaded.txt

Testing New Features Since Prototype:

Authentication – this feature is tested by the fact that to get any of the above responses from our honeypot the client machine must include the username “tomcat” and the password “tomcat” in every request to the honeypot. If the username/password is not provided or an incorrect username or password is provided the honeypot with respond with an HTTP status code 401. Upon providing the correct username and password, the client will be served by our honeypot if the client’s IP address is not found in the honeypot’s blacklist log file.

Blacklist feature – once a Client uploads a file to the honeypot, any further requests from the client will result in a response from the honeypot with an HTTP status code of 403 Forbidden or the honeypot will ignore the request. To test this feature, follow the steps in Test case 3 above and after successfully uploading a file attempt to continue to interact with the honeypot from the same client. The honeypot will prevent the client from continuing to interact with the honeypot. To reenable the client’s ability to interact with the honeypot, the client’s IP address must be deleted from the blacklist log file.

Multi-threading feature – To test the ability to serve multiple clients, try any of the above test cases simultaneously from different client VMs or machines

Wider range of files to upload – For the curl commands above, the file path after the -T parameter can be to any file the user chooses to upload. The honeypot, upon receiving the PUT request from the client with the specified file, will save the contents of the file to a text file with the filename in the form timeofupload_IPaddressOfClient.txt.

Linux Command Line Utilities Explained

Wget:

The wget utility is a Linux command line utility used to send HTTP requests to a HTTP server from the Linux command line. The format for a proper wget command is as follows:

$ wget URI_of_network_resource_you_are_requesting wget_parameters

A URI for HTTP looks like this:

http://[[username]:[password]@]hostname[:port]/path[?query]

For our project an example URI would be:

http://tomcat:tomcat@IPAddressOFHoneypot:8080/manager/text/serverinfo

In our test cases above we use several parameters in wget: -O - and -q

- -O - tells wget to print the contents of the file received from the server to STDOUT
- -q suppresses wget's status messages which show the size of the HTTP request as well as the HTTP status code of the response from the server.

cURL

We use cURL in our test cases above to upload a file to our honeypot. This was done because to upload a file, one must use an HTTP PUT request and wget appears to only allow GET requests.

An example cURL command is as follows:

curl -T test.txt --user tomcat:tomcat -X PUT
http://localhost:8080/manager/text/deploy?path=/test

Parameters Explained:

- -T: after the "-T" enter the path to the file that the user wishes to upload to the honeypot

- --user: after “--user” the username and password for the honeypot is entered in the format username:password. For our honeypot it is tomcat:tomcat
- -X: the “-X” option allows the user to specify the type of HTTP request to send to the server and the URI of that request. To upload a file to the server a PUT request must be used. For our honeypot enter after the “-X”: http://IPADDRESSHONEYPOT:8080/manager/text/deploy?path=/PATH_TO_SEND_FILE_TO.
  The PATH_TO_SEND_FILE_TO portion of the above does not matter as our honeypot is hard coded to save the file uploaded to the honeypot to a specific location. This is done to prevent the attacker from uploading files to undesirable locations on the honeypot’s host system’s hard drive.

Nmap:

Nmap is an open source network scanning and mapping application that is widely used in the IT industry. For our project, we designed our Tomcat Honeypot to identify itself to nmap as Apache Tomcat. The nmap command used to test this functionality is:

nmap -sS -sV <IPADDRESSofTOMCATSYSTEM>

Parameters Explained:

- -sS: tells nmap to scan the most common ports for TCP connections
- -sV: tells nmap to try to identify what services the system it is scanning is running on any ports that it finds open on the target system

Execution Instructions:

Import the provided FINAL_TomcatHoneyPot.zip file to NetBeans IDE and execute the code on the host machine.