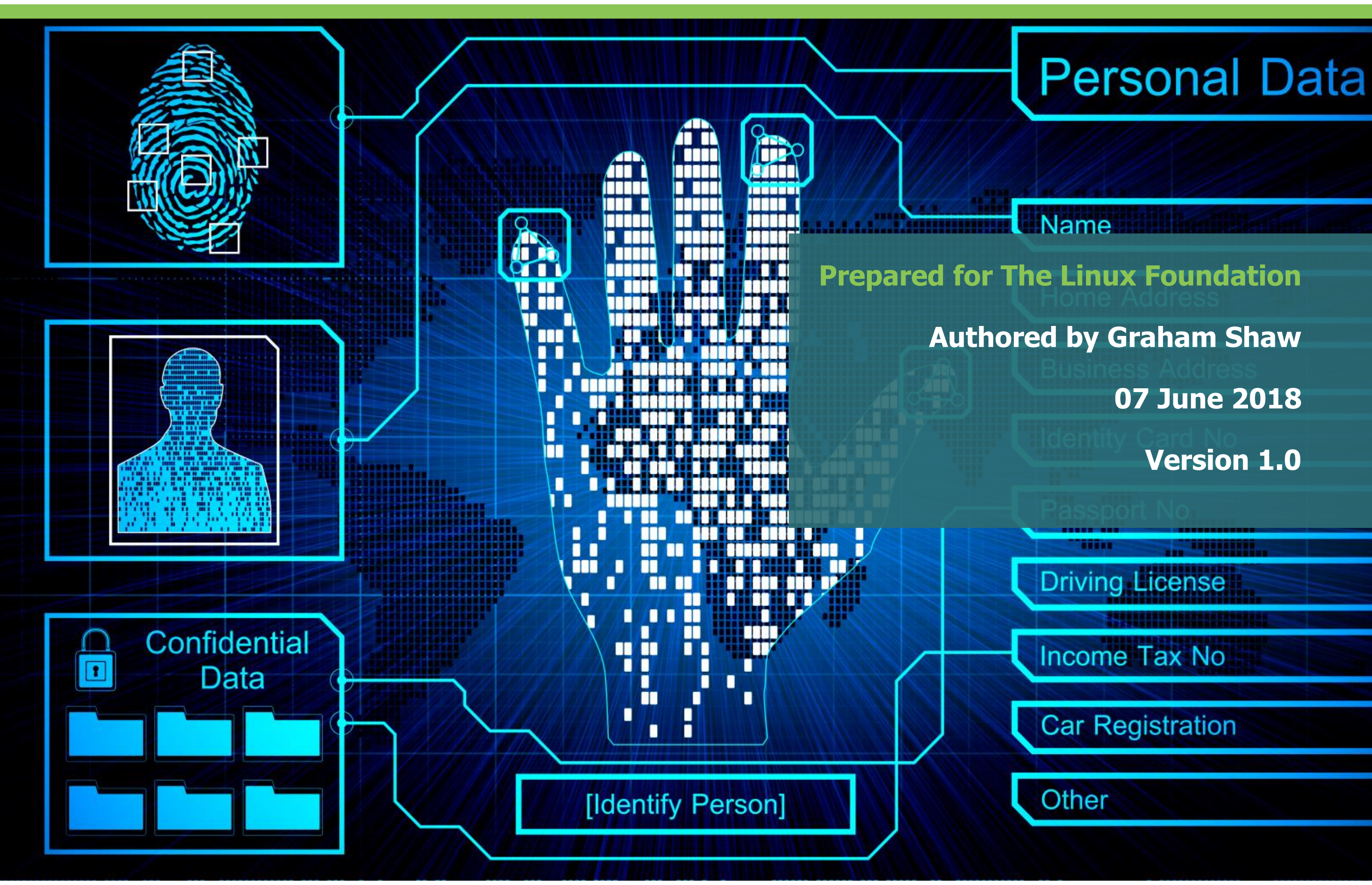
NETTITUDE

INTELLIGENT CYBER SECURITY & RISK MANAGEMENT

Penetration Test Technical Report



Prepared for The Linux Foundation

Authored by Graham Shaw

07 June 2018

Version 1.0

NETTITUDE

INTELLIGENT CYBER SECURITY & RISK MANAGEMENT

Penetration Test Technical Report

Nettitude provides a wealth of knowledge, expertise and experience in regards to Data Security. We provide comprehensive vulnerability assessment, penetration testing and application assessment services. Our team of dedicated security consultants deliver best in class testing capability as well as strong remediation advice and guidance.

NETTITUDE

REPORT CONTENTS

1 Distribution List 4

1.1 Nettitude 4

1.2 The Linux Foundation 4

1.3 Revision History 4

2 Engagement Particulars 5

2.1 Background 5

2.2 Rules of Engagement 5

2.3 Scope 5

2.4 Testing Window Observations / Constraints 5

3 Findings 6

3.1 Software Security Assessment 6

4 Analysis: Software Security Assessment 7

4.1 Medium: Credentials stored in world-readable files 7

4.2 Medium: Playground server accepts connections from any IP address by default 8

4.3 Low: REST server allows connections using insecure TLS cipher suites 9

4.4 Low: REST server allows connections using older TLS protocol versions 11

4.5 Information Leakage: REST API discloses unnecessary information in error messages 12

NETTITUDE

1 DISTRIBUTION LIST

1.1 Nettitude

Name	Title
Graham Shaw	Security Consultant
Miles Corn	Account Manager
Iain Wallace	Security Consultant

1.2 The Linux Foundation

Name	Title
David Huseby	Security Maven, Hyperledger

1.3 Revision History

Version	Issue Date	Issue By	Comments
0.1	25 May 2018	Graham Shaw	Initial Draft
0.2	01 June 2018	Iain Wallace	Quality Assurance
0.3	04 June 2018	Miles Corn	Quality Assurance
1.0	07 June 2018	Graham Shaw	Final

2 ENGAGEMENT PARTICULARS

2.1 Background

This report serves as the technical documentation for the recent security assessment performed for The Linux Foundation by Nettitude. For a high level assessment of the tested environment, please refer to the associated management report:

- MANAGEMENT_REPORT_Penetration_Test_The_Linux_Foundation_May_2018_v1.0.pdf.

2.2 Rules of Engagement

The penetration test was performed in line with the following rules of engagement:

- Nettitude's product testing methodology was used.
- Social engineering was not applicable.
- The software was installed on equipment under the control of Nettitude for testing. Testing of systems belonging to The Linux Foundation was not permitted.
- The testing and reporting was permitted and performed during a 15 day period; 01 May to 24 May 2018. Any results held in this report relate to the status of the tested software on those dates.

2.3 Scope

Nettitude were tasked with performing a penetration test with the following scope:

Name	Source
Hyperledger Composer	https://github.com/hyperledger/composer

2.4 Testing Window Observations / Constraints

The time frame provisioned for the completion of this engagement was adequate.

No constraints were encountered during the engagement.

3 FINDINGS

3.1 Software Security Assessment

Description	Severity	Ease of Exploitation	Recommendation	Reference
Credentials stored in world-readable files	**Medium**	**Trivial**	Avoid making sensitive files world-readable.	4.1
Playground server accepts connections from any IP address by default	**Medium**	**Trivial**	Bind to the loopback address by default.	4.2
REST server allows connections using insecure TLS cipher suites	**Low**	**Complex**	Disable weak cipher suites.	4.3
REST server allows connections using older TLS protocol versions	**Low**	**Complex**	Consider disabling older versions of TLS.	4.4
REST API discloses unnecessary information in error messages	**Information Leakage**	**Trivial**	Avoid extraneous information in error messages	4.5

4 ANALYSIS: SOFTWARE SECURITY ASSESSMENT

4.1 Medium: Credentials stored in world-readable files

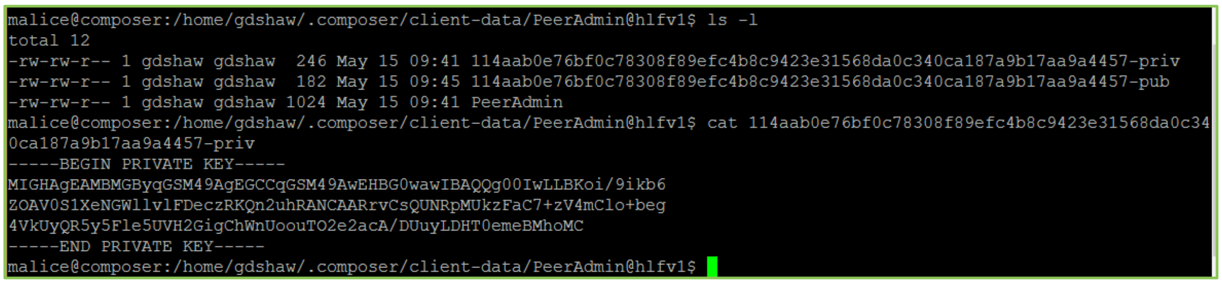
Description of the Issue

Composer stores data in a directory named "~/.composer". Within this is a subdirectory named "client-data" containing administrative credentials for accessing Fabric networks. These credentials are world-readable.

For example, the test account "PeerAdmin@hlfv1" (which can be created as part of the development environment) has an associated subdirectory "~/.composer/client-data/PeerAdmin@hlfv1". Within this were found:

- A pair of files containing the public-private keypair for this identity.
- A further file called "PeerAdmin" in JSON format. Amongst other information, this contains a field for recording the enrollment secret if one has been used.

Nettitude was able to demonstrate access to this data from an unrelated account on the same machine:



```
malice@composer:/home/gdshaw/.composer/client-data/PeerAdmin@hlfv1$ ls -l
total 12
-rw-rw-r-- 1 gdshaw gdshaw  246 May 15 09:41 114aab0e76bf0c78308f89efc4b8c9423e31568da0c340ca187a9b17aa9a4457-priv
-rw-rw-r-- 1 gdshaw gdshaw  182 May 15 09:45 114aab0e76bf0c78308f89efc4b8c9423e31568da0c340ca187a9b17aa9a4457-pub
-rw-rw-r-- 1 gdshaw gdshaw 1024 May 15 09:41 PeerAdmin
malice@composer:/home/gdshaw/.composer/client-data/PeerAdmin@hlfv1$ cat 114aab0e76bf0c78308f89efc4b8c9423e31568da0c34
0ca187a9b17aa9a4457-priv
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQg00IwLLBKoi/9ikb6
ZOAV0S1XeNGWllvlFDeczRKQn2uhRANCAARrvCsQUNRpMUkzFaC7+zV4mClo+beg
4VkUyQR5y5Fle5UVH2GigChWnUoouTO2e2acA/DUuyLDHT0emeBMhoMC
-----END PRIVATE KEY-----
malice@composer:/home/gdshaw/.composer/client-data/PeerAdmin@hlfv1$
```

Figure 1: World-readable credentials

(Interestingly, essentially the same data appears to be duplicated within the 'cards' subdirectory, where it has been appropriately protected.)

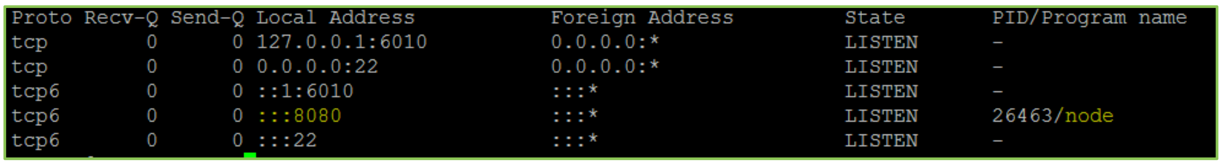
Nettitude Recommends

1. Change the file modes for these files to prevent them from being world-readable.
2. Consider doing likewise for all directories and files from "~/.composer" downwards, unless there is a good reason why they should remain world-readable.

4.2 Medium: Playground server accepts connections from any IP address by default

Description of the Issue

By default, the composer-playground server binds to port 8080 on the IPv6 wildcard address (:::8080), which in principle means that it will accept IPv6 or IPv4 connections from anywhere on the Internet. In practice, it will typically be located behind a firewall or NAT gateway which will block connections from the public Internet, however even if this is assumed, it could potentially be exposed to a large number of machines on a local area network.



```
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:6010          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp6       0      0 ::1:6010                :::*                    LISTEN      -
tcp6       0      0 :::8080                 :::*                    LISTEN      26463/node
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

While this certainly could be what the user requires and intends to happen, there is the alternative and perhaps more likely possibility that the playground will only be accessed by a web browser on the same host. In such circumstances it is good practice to bind to the loopback interface by default, allowing access via other interfaces if and only if the user has explicitly instructed the server to do that.

In the absence of authentication (which was presumably a design decision rather than an oversight), and with the server bound to the wildcard address, an attacker able to connect to the playground server would be able to:

- Use business network cards held by the server to connect and interact with Fabric networks using the composer-playground user interface.
- Download business network cards, including the private keys and enrollment secrets contained within them, in order to connect and interact with Fabric networks independently of composer-playground.

Nettitude was able to demonstrate the latter, creating a pair of cards on one machine for interacting with a Fabric network, then downloading those cards from a second machine.

Nettitude Recommends

1. Have the playground server bind to the loopback address by default.
2. Reject requests to the loopback address range if a hostname has been specified which is not equal to "localhost", to prevent DNS rebinding attacks
3. Allow the user to bind to a different address if preferred (which could be the wildcard address).

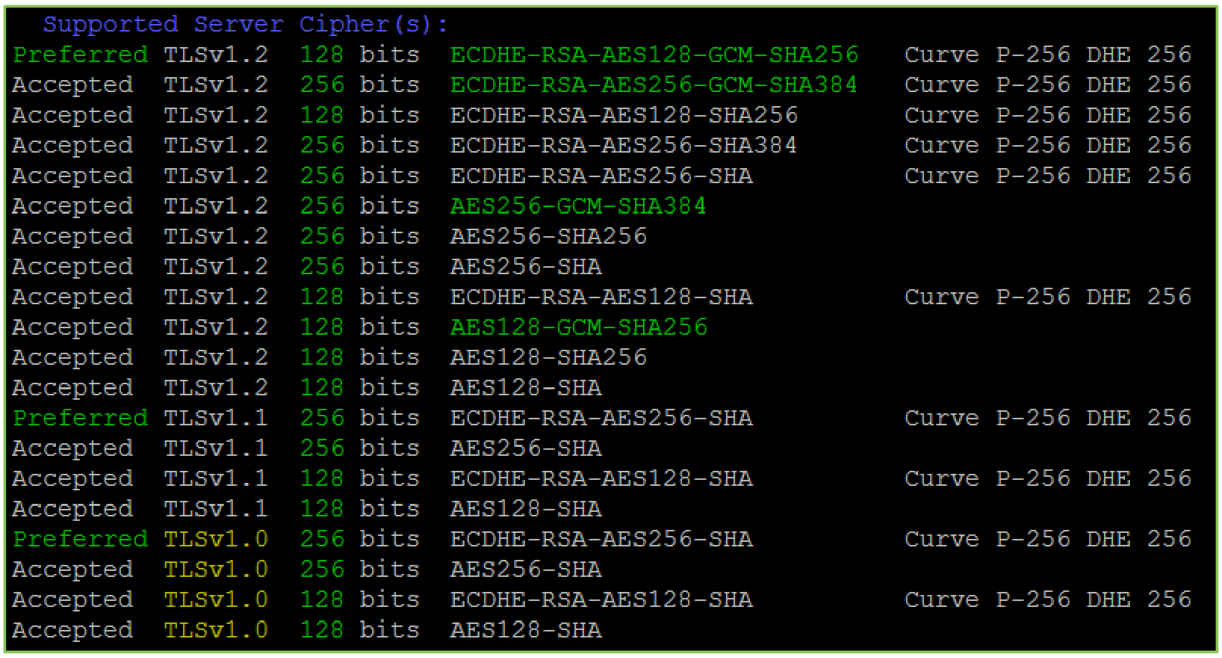
A case could be made for doing likewise in the case of the REST server, and for any other sockets which are opened for listening, however this can be argued either way due to the REST server being both more secure, and more likely to receive legitimate connections from other machines.

4.3 Low: REST server allows connections using insecure TLS cipher suites

Description of the Issue

When the REST server is configured to accept TLS connections, it allows clients to choose one of twelve supported cipher suites. Some of these (including the default cipher for TLS v1.2) provide very good security, and none of them are very weak, however the configuration as it stands is not optimal from a security perspective.

The supported cipher suites are as follows:



```
  Supported Server Cipher(s):
Preferred TLSv1.2  128 bits  ECDHE-RSA-AES128-GCM-SHA256   Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-GCM-SHA384   Curve P-256 DHE 256
Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA256       Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA384       Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA          Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  AES256-GCM-SHA384
Accepted  TLSv1.2  256 bits  AES256-SHA256
Accepted  TLSv1.2  256 bits  AES256-SHA
Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA          Curve P-256 DHE 256
Accepted  TLSv1.2  128 bits  AES128-GCM-SHA256
Accepted  TLSv1.2  128 bits  AES128-SHA256
Accepted  TLSv1.2  128 bits  AES128-SHA
Preferred TLSv1.1  256 bits  ECDHE-RSA-AES256-SHA          Curve P-256 DHE 256
Accepted  TLSv1.1  256 bits  AES256-SHA
Accepted  TLSv1.1  128 bits  ECDHE-RSA-AES128-SHA          Curve P-256 DHE 256
Accepted  TLSv1.1  128 bits  AES128-SHA
Preferred TLSv1.0  256 bits  ECDHE-RSA-AES256-SHA          Curve P-256 DHE 256
Accepted  TLSv1.0  256 bits  AES256-SHA
Accepted  TLSv1.0  128 bits  ECDHE-RSA-AES128-SHA          Curve P-256 DHE 256
Accepted  TLSv1.0  128 bits  AES128-SHA
```

The concerns that this raises are as follows:

- Cipher suites which use RSA for key exchange. These do not provide perfect forward secrecy, meaning that if the private key is ever compromised, it is possible to decrypt messages retrospectively.

- Cipher suites which use the CBC (Cipher Block Chaining) cipher mode. In combination with TLS v1.0, these are vulnerable to the BEAST attack. Even if TLS v1.0 is not being used, use of GCM is preferable to CBC due to the former being an authenticated cipher mode.

- Cipher suites which use SHA-1 as the hashing algorithm. SHA-1 is no longer considered to be a fully secure hashing algorithm, particularly following the announcement of the first public collision in February 2017. It remains acceptably secure for general use in TLS cipher suites, due to the relatively short time window available for mounting an attack, however it is not the optimal choice.

Although security is best served by disabling the affected cipher suites, there is a trade-off to be made between security and compatibility. All of the listed algorithms are widely used on the public Internet, including by e-commerce and even banking websites. None of them are so bad that merely permitted (as opposed to requiring) their use would be likely to attract serious criticism in the context of Hyperledger Composer.

Nettitude Recommends

1 Consider disabling support for cipher suites which use RSA for key exchange. These are the ones which do not begin with the prefix ECDHE. It is immaterial whether RSA appears in the name.

2 Consider disabling cipher suites which use the CBC (Cipher Block Chaining) cipher mode. These are the ones which do not contain GCM in the name.

3 Consider disabling cipher suites which use SHA-1 as the hashing algorithm. These are the ones which end with the suffix SHA, as opposed to SHA256 or SHA384.

4.4 Low: REST server allows connections using older TLS protocol versions

Description of the Issue

When the REST server is configured to accept TLS connections, it allows clients to connect using versions 1.0 and 1.1 of the TLS protocol. By itself this is not a vulnerability, since at the time of writing there were no known methods which allowed the security of a TLS connection to be broken merely due to it using one of these protocol versions. However, they do lack some of the security features provided by more recent versions of TLS, and for this reason allowing their use would not be considered security best practice (for version 1.0 especially, and to a lesser extent for version 1.1).

The server does also support TLS version 1.2, and provided this is also support by the client then this ought to be what is negotiated and used. The risks which result from allowing the use of older versions of the protocol are that:

- While the choice of client is primarily the prerogative of the end user, the organization running the server may have business, ethical or legal reasons for wanting to prevent end users from putting security at risk.
- SSL, the predecessor protocol on which TLS is based, can in some circumstances be exploited by means of a downgrade attack, whereby a malicious third party causes the client and server to negotiate a less secure version of the protocol than they would otherwise be capable of using. The possibility of an attack method of this nature being discovered against TLS cannot be excluded, but would be very unlikely to work if older protocol versions have been disabled.

Although security is best served by disabling TLS versions 1.0 and 1.1, there is a trade-off to be made between security and compatibility:

- Use of TLS v1.1 is defensible for most websites which need to be accessible by the general public, unless there is a strong reason for prioritizing security over accessibility.
- Use of TLS v1.0 is defensible for public websites where security is of low importance, such as typical informational websites, however it would imprudent to use it for protecting sensitive personal or financially valuable information. Note in particular that it is no longer considered strong enough for protecting cardholder data on e-commerce sites which need to be PCI DSS compliant.

At the time of writing, TLS v1.3 had been finalized as a draft standard, and was close to being formally released. It will be some time before it is appropriate to recommend disablement of version 1.2 for most types of server, however it would be good practice to enable version 1.3 if (or failing that, when) a suitable implementation is available.

Nettitude Recommends

1 Disable TLS v1.0, unless you wish to strongly prioritize browser compatibility over security.

2 Disable TLS v1.1, if you wish to strongly prioritize security over browser compatibility.

3 Enable TLS v1.3 as soon as it is reasonably feasible to do so.

4.5 Information Leakage: REST API discloses unnecessary information in error messages

Description of the Issue

When a request is made to the REST server for an unrecognized URL, the response is typically similar in form to the following:

```
{"error":{"statusCode":404,"name":"Error","message":"Cannot GET /foo","status":404,"stack":"Error: Cannot GET /foo\n    at
raiseUrlNotFoundError (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-
server/node_modules/loopback/server/middleware/url-not-found.js:21:17)\n    at Layer.handle [as handle_request]
(/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/layer.js:95:5)\n    at
trim_prefix (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-
server/node_modules/express/lib/router/index.js:317:13)\n    at /home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-
server/node_modules/express/lib/router/index.js:284:7\n    at Function.process_params
(/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/index.js:335:12)\n    at
next (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/index.js:275:10)\n
at /home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/index.js:635:15\n    at
next (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/index.js:260:14)\n
at Function.handle (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-
server/node_modules/express/lib/router/index.js:174:3)\n    at router (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-
rest-server/node_modules/express/lib/router/index.js:47:12)\n    at Layer.handle [as handle_request]
(/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/layer.js:95:5)\n    at
trim_prefix (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-
server/node_modules/express/lib/router/index.js:317:13)\n    at /home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-
server/node_modules/express/lib/router/index.js:284:7\n    at Function.process_params
(/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/index.js:335:12)\n    at
next (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/express/lib/router/index.js:275:10)\n
at urlencodedParser (/home/gdshaw/.nvm/versions/node/v8.11.1/lib/node_modules/composer-rest-server/node_modules/body-
parser/lib/types/urlencoded.js:91:7)"}}
```

Note the disclosure of:

- The absolute pathname of the directory within which the software is located.
- The installed version of Node.js

In this instance, one of the pathname components matches the username of the account being used to run the server. This will not always be the case, but can be expected to happen frequently. Technically, it could be characterized as leakage of personally identifiable information.

While this information is unlikely to be of significant value to an attacker as an end in itself, it could potentially be of value for planning and mounting other attacks. For example:

- Knowledge of where the software is installed could be helpful for writing malicious payloads.
- Knowledge of the Node.js version would allow an attacker to check for publicly-known vulnerabilities in that library.

(These error messages also disclose information about the internals of the software, however this is not a security concern since the software is Open Source.)

Nettitude Recommends

1 Avoid disclosure of extraneous information in error messages, unless explicitly requested by the server administrator.