SSD **Name:** _____
Winter Term 2021
Exam 1 **Immatriculation Number:** _____
17.12.2021
Time Limit: 90 Minutes **SSD Challenge Username:** _____

This exam contains 24 pages (including this cover page) and 5 questions. Check to see if any pages are missing. Enter all requested information on the top of this page, and put your immatriculation number on the top of every page, in case the pages become separated.

You may *not* use your books, notes, or any calculator on this exam.

- Write all your answers on these sheets!
- Write legible - illegible answers are considered wrong.
- If you need more space, use the back of the pages; clearly indicate when you have done this.

Do not write in the table to the right.

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	5	
Total:	45	

Bonus Lecture Challenges:

Bonus	4.5	

1. (10 points) **Low-Level Security Trivia**

   There are 10 statements. For each statement, check the correct answer. **Note:** a *wrong* answer results in -1 point, a *correct* answer in +1 point, and *no answer* in 0 points for the question. However, you cannot get below 0 points in total. There is always one correct answer.

C++ virtual functions use...

- ◯ vmaps
- ◯ vlists
- ◯ vvector
- ◯ vtables

Where is a local buffer typically allocated?

- ◯ Heap
- ◯ Stack
- ◯ Shared Memory
- ◯ Data

Which type of overflow is not a typical bug?

- ◯ Integer overflow
- ◯ Heap overflow
- ◯ Stack overflow
- ◯ Pointer overflow

Which exploitation technique does not exist?

- ◯ ret2libc
- ◯ PROP
- ◯ SROP
- ◯ ROP

What is part of an ELF binary?

- ◯ File Header Table
- ◯ Data Header Table
- ◯ Code Header Table
- ◯ Program Header Table

What is a spatial memory safety violation?

- ◯ Use-After-Free
- ◯ Buffer overflow
- ◯ Use Of Uninitializd Memory
- ◯ Null pointer dereference

What is important for type coercion?

- ◯ Sign
- ◯ Rank
- ◯ Alignment
- ◯ Size

What does GOT stand for?

- ◯ Generic Offset Translation
- ◯ Global Offset Table
- ◯ Global Offcore Trace
- ◯ Generic Opcode Table

In which direction are arguments passed on the stack in the cdecl calling convention?

- ◯ Left to right
- ◯ Right to left
- ◯ Arbitrary
- ◯ They are in registers

What is the numeric value of `PROT_EXEC|PROT_READ|PROT_WRITE`?

- ◯ 0
- ◯ 3
- ◯ 5
- ◯ 7

2. (10 points) **Countermeasures**

   (a) (3 points) Illustrate how canaries work.

   (b) (3 points) What are the advantages and disadvantages of W $\oplus$ X compared to canaries?

   (c) (4 points) Describe a countermeasure for stack exploitation that works in a case where both canaries and W $\oplus$ X fail. Describe the attack and how it is mitigated in specific.

3. (10 points) **Finding bugs**

   (a) (3 points) What is reverse engineering and explain two non-malicious applications for reverse engineering.

   (b) (3 points) Name the two classes of reverse-engineering tools and explain their differences.

   (c) (4 points) Explain how binary diffing works and describe the most important usecase.

4. (10 points) **Exam Hacklets**

   To get points for the exam hacklets, you have to provide your SSD CTF username and tick the hacklets you solved.
   You do not have to answer hacklets which you already solved and submitted online.

   SSD CTF username: _____

   (a) (0.5 points) ☐ *(Solved in take home exam part) Hacklet #1 (fmt_string)*
   How can you crash a program with a format string? How can format string vulnerabilities be mitigated?

   (b) (0.7 points) ☐ *(Solved in take home exam part) Hacklet #2 (Birds)*
   Explain how you can open a shell by analyzing the following snippet:

```
extern int open_shell();

void func() {
  char buffer[512];
  fgets(buffer, sizeof(buffer) - 1, stdin);
  printf(buffer);
  write(STDOUT_FILENO, "\n", 1);
}

int main() {
  func();
  return 0;
}
```

(c) (0.7 points) ☐ *(Solved in take home exam part) Hacklet #3 (Book Store)*
Which type of bug was triggered in Book Store? Can you sketch the exploit chain by looking at the following snippet?

```
struct Book {
  char name[8];
  char* description;
  size_t cost;
  struct Book* next;
};
struct Book* head;
void add() {
  struct Book* book = malloc(sizeof(struct Book));
  memset(book->name, 0, 8);
  scanf("%7s", book->name);
  book->description = malloc(32);
  int n = read(0, book->description, 31);
  if (n < 0) return;
  book->cost = 1;
  add_to_list(head, book);
}
void delete() {
  int i;
  scanf("%d", &i);
  if (head == 0) return;
  struct Book* cur = get_i_of_list(head, i);
  if (!cur) return;
  free(cur->description);
  memset(cur->name, 0, 16);
  cur->cost = 0;
}
void change() {
  int i;
  scanf("%d", &i);
  if (head == 0) return;
  struct Book* cur = get_i_of_list(head, i);
  if (!cur) return;
  memset(cur->description, 0, 32);
  int n = read(0, cur->description, 31);
```

```
  if (n < 0) return;
}
void checkout() {
  size_t cost = sum_cost_of_list(head);
  if (cost = 0xdeadbeef) { system("/bin/sh"); }
  exit();
}
int main() {
  for (size_t i = 0; i < 0x100; ++i) {
    int choice;
    scanf("%d", &choice);
    if (choice == 0) add();
    else if (choice == 1) delete();
    else if (choice == 2) change();
    else if (choice == 3) checkout();
    else puts("invalid command");
  }
}
```

(d) (0.7 points) ☐ *(Solved in take home exam part) Hacklet #4 (Knock-kock)*
Sketch the exploit change and locate the bug in the following program.

```
void flag() { system("cat flag.txt"); }
void knock() {
  struct {
    int cnt;
    char input[4];
    int index;
    char chooser[4];
  } data;

  memset(&data, 0, sizeof(data));
```

```
  while (data.cnt++ < 4) {
    printf("Enter content for index %d:\n", data.index);
    if (fgets(data.input, 10, stdin) == 0 || data.input[0] == '\n') {
      puts("Unable to read input.");
      return;
    }
    data.chooser[data.index++] = data.input[0];
  }

  srand(data.chooser[0] ^ data.chooser[1] ^ data.chooser[2] ^ data.
      chooser[3]);
  int choice = rand();
  printf("Your choice was: %d\n", choice);
}

int main() {
  knock();
  return 0;
}
```

(e) (0.7 points) ☐ *Perfect encryption*
Why was the encryption not perfect in that example? Provide an input that triggers the bug and sketch a fix.

```
char secret[16];
extern void readSecret(char* secret);
int checkKey(const char* key, int length)
{
  int zeros = 0;
  for (int i = 0; i < length; i++)
    if (key[i] == 0)
      if (zeros++ > 1) return 1;
  return 0;
}
int encrypt(char* state, const char* key, int length) {
  char temp[length];
  int ok = 1;
  memcpy(temp, state, length);
  for (int i = 0; i < strlen(key); i++) {
    state[i] = temp[i] ^ key[i];
    ok &= temp[i] == state[i];
  }
  return !ok;
}
void func() {
  int fd = open("./user_controlled.key", O_RDONLY);
  char buffer[16];
  int ret_value;
  ret_value = read(fd, buffer, 16);
  if (ret_value != 16) exit(-1);
  ret_value = checkKey(buffer, 16);
  if (ret_value != 0) exit(-1);
  ret_value = encrypt(secret, buffer, 16);
  if (ret_value != 0) exit(-1);
  printf("secret: %s\n", secret);
}
int main() {
  readSecret(secret);
  func();
}
```

(f) (0.7 points) ☐ *(Solved in take home exam part) Hacklet #6 (ROP me)*
Sketch the ROP chain for this hacklet.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <stdlib.h>
#include <unistd.h>
#include <assert.h>
void success(size_t arg)
{
  if (arg != 0xbadaffe) exit(-2);
  FILE* f = fopen("./flag.txt", "r");
  char c;
  while ((c = fgetc(f)) != EOF) printf("%c",c);
  fclose(f);
}
void ropMe()
{
  size_t buffer;
  read(0, &buffer, sizeof(size_t)*8);
  if (*(size_t*)(&buffer+4) >= (size_t)&success && *(size_t*)(&buffer+4)
      < (size_t)&ropMe) exit(-1);
}
void main()
{
  printf("I bet you, cannot reach %p\n", &success);
  ropMe();
  exit(0);
}
```

(g) (0.7 points) ☐ *(Solved in take home exam part) Hacklet #7 (Safe Encryption)*
Sketch the problem, exploit and fix of this hacklet. Which files are interesting to read on a Linux system?

```
extern int check_permission(const char* file_name, gid_t egid);

extern int read_data(const char* file_name, char* buffer, int length);
void vuln(gid_t egid)   {
  const char* file_name = "./secret.key"
  int ret_value;
  char buffer[64];
  memset(buffer, 0, 64);
  ret_value = check_permission(file_name, egid);
  if (ret_value != 0) exit(-1);
  ret_value = read_data(file_name, buffer, 63);
  if (ret_value != 0) exit(-1);
  ret_value = check_permission(file_name, egid);
  if (ret_value != 0) exit(-1);
  printf("content: %s\n", buffer);
}
```

(h) (0.7 points) ☐ *(Solved in take home exam part) Hacklet #8 (Sorted)*
Write an x86-64 shellcode snippet that opens the flag.txt (file size is 20 bytes) file and prints the content.

(i) (1.0 points) ☐ *(Solved in take home exam part) Hacklet #9 (Dealership)*
Sketch an exploit and a fix for this vulnerable snippet.

```
class Vehicle {
private:
  virtual void flag() { system("cat flag.txt"); }
protected:
  std::string name;
  Vehicle(std::string name) { this->name = name; }
public:
  virtual ~Vehicle() { }
  virtual void drive() = 0;
};

class Car: public Vehicle {
protected:
  int seats;
public:
  Car(std::string name, int seats) : Vehicle(name) { this->seats = seats;
       }
  virtual void drive() {
    std::cout << "Seems like a nice car you got there." << std::endl;
  }
};

Vehicle *garage[5];

int main(int argc, char *argv[]) {
  // create a car ...
  garage[0] = new Car("SuperNiceCar", 4);

  // ... and delete it again
  delete garage[0];

  // lets get some fuel
  std::cout << "Where do we get the fuel?" << std::endl;
  int len = sizeof(Car);
  void *fuel = new char[len];
  read(open("payload", O_RDONLY), fuel, len);
  std::cout << "Fuel is filled up again!" << std::endl;

  // what could go wrong?
  garage[0]->drive();
}
```

(j) (1 point) ☐ *(Solved in take home exam part) Hacklet #10 (Heart bleed)*
Locate the bug in the following snippet and discuss potential attacks.

```
typedef struct __attribute__((packed)) user  {
  char is_registered;
  char is_admin;
  char password[21];
  char username[11];
  size_t info_length;
  char info[21];
} user;

#define MAX_USERS 256
uint8_t reg_counter = 0;
int8_t logged_in_user = -1;
user registered_users[MAX_USERS];

/**
 * @brief prints flag if logged in and registered_users[logged_in_user].
     is_admin is 1
 */
extern void flag()
/**
 * @brief lists all users ID (=index), admin flag, username and info (using
      specified info_length)
 */
extern void list_users();
/**
 * @brief asks for username/password, sets logged_in_user to user ID (=
     index) if user found
 */
extern void login();
/**
```

```
 * @brief asks for username/password/info of correct lengths, calls
     add_user(0, pass, user, info)
 */
extern void register_user();

void logout() { logged_in_user = -1; }

void add_user(char is_admin, char *password, char *username, char *info) {
  registered_users[reg_counter].is_registered = 1;
  registered_users[reg_counter].is_admin = is_admin;
  strncpy(registered_users[reg_counter].password, password, 20);
  strncpy(registered_users[reg_counter].username, username, 10);
  strncpy(registered_users[reg_counter].info, info, 20);
  registered_users[reg_counter].info_length = strlen(registered_users[
      reg_counter].info);

  reg_counter++;
}

void rename_user() {
  if (logged_in_user == -1) return;
  puts("Enter new username:");
  fgets(registered_users[logged_in_user].username, 0x10, stdin);
  registered_users[logged_in_user].username[strcspn(registered_users[
      logged_in_user].username, "\n")] = 0;
}

int main() {
  memset(registered_users, 0, sizeof(user) * MAX_USERS);
  char secret_password[3][16] = ... // super-secure passwords
  add_user(0, secret_password[0], "guest", "i'm nobody");
  add_user(1, secret_password[1], "admin", "i'm the boss");
  add_user(0, secret_password[2], "test", "don't mind me");

  printf("Welcome to our service! Choose your action.\n");
  printf("btw, we don't need to hash our passwords because our service is
      very secure.\n");

  while (1) {
    line_buffer = ... // read line from stdin
    if (strncmp(line_buffer, "list", 4) == 0) list_users();
    else if (strncmp(line_buffer, "login", 5) == 0) login();
    else if (strncmp(line_buffer, "logout", 6) == 0) logout();
    else if (strncmp(line_buffer, "register", 8) == 0) register_user();
    else if (strncmp(line_buffer, "rename", 6) == 0) rename_user();
    else if (strncmp(line_buffer, "flag", 4) == 0) flag();
    else if (strncmp(line_buffer, "exit", 4) == 0) break;
  }

  return 0;
}
```

(k) (1.2 points) ☐ *(Solved in take home exam part) Hacklet #11 (Echo)*
Explain how you can leak stack canaries. How can you leak the libc base address? (Provide a sketch)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
void respond() {
  char response[100];
  int n = 0;
  memset(response, 0, 0x100);
  n = read(0, response, sizeof(response));
  if (n < 0) return;
  else if (n > 0 && response[n-1] == '\n') response[n-1] = 0;
  write(1, response, sizeof(response));
}
void doStuff() {
  do   {
    printf("> ");
    fflush(stdout);
    respond();
    printf("> done\n");
  } while (1);
}
int main() {
  while (1) {
    int pid = fork();
    if (pid == 0) doStuff();
    else wait(NULL);
  }
}
```

(l) (1.4 points) ☐ *(Solved in take home exam part) Hacklet #12 (Garden)*
Leak the libc base and and spawn a shell.

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

char colors[][8] = {
  "red",
  "green",
  "blue",
  "grey",
  "yellow",
  "white",
  "orange"
};

struct Vuln {
  char name[32];
  size_t address;
  size_t value;
};

static char* vuln1() {
  int choice;
  for (int i = 0; i < 7; i++)
  {
    printf("%d: %s\n", i, colors[i]);
  }
  printf("color: ");
  fflush(stdout);
  scanf("%d", &choice);
  if (choice >= 7)
  {
    return 0;
  }
  return colors[choice];
}
static void vuln2() {
  struct Vuln* vuln = calloc(1, sizeof(struct Vuln));
  printf("name: ");
  fflush(stdout);
  scanf("%31s", vuln->name);
  printf("address: ");
  fflush(stdout);
  scanf("%ld", vuln->address);
  printf("value: ");
  fflush(stdout);
  scanf("%ld", vuln->value);
  *(size_t*)vuln->address = vuln->value;
  free(vuln);
}
int main() {
  char* color = vuln1();
  printf("color: %s\n");
  vuln2();
}
```

5. (5 points) **(Bonus) Lecture Challenges**

   To get points for the lecture challenges, you have to provide your SSD CTF username and answer a short question for every lecture challenge you have solved.

   (a) (0.5 points) *Challenge #1 (minielf)*
   With which tool did you create the ELF binary?

   (b) (0.5 points) *Challenge #2 (quadfloat)*
   How many bits does a IEEE 754 quadruple-precision binary floating-point number have?

   (c) (0.5 points) *Challenge #3 (needle)*
   Which git command did you use to solve the challenge?

   (d) (0.5 points) *Challenge #4 (heart-starty)*
   Which technique did you use to reveal the flag?

   (e) (0.5 points) *Challenge #5 (secwrap)*
   Which function is used to apply all the seccomp rules?

   (f) (0.5 points) *Challenge #6 (aslr)*
   Name one compiler flag which has an effect on ASLR.

Appendix: ASCII Table

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL (null)	0x20	32	space	0x40	64	@	0x60	96	‘
0x01	1	SOH (start of heading)	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX (start of text)	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX (end of text)	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT (end of transmission)	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ (enquiry)	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK (acknowledge)	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL (bell)	0x27	39	’	0x47	71	G	0x67	103	g
0x08	8	BS (backspace)	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB (horizontal tab)	0x29	41	)	0x49	73	I	0x69	105	i
0x0a	10	LF (new line)	0x2a	42	*	0x4a	74	J	0x6a	106	j
0x0b	11	VT (vertical tab)	0x2b	43	+	0x4b	75	K	0x6b	107	k
0x0c	12	FF (form feed)	0x2c	44	,	0x4c	76	L	0x6c	108	l
0x0d	13	CR (carriage return)	0x2d	45	-	0x4d	77	M	0x6d	109	m
0x0e	14	SO (shift out)	0x2e	46	.	0x4e	78	N	0x6e	110	n
0x0f	15	SI (shift in)	0x2f	47	/	0x4f	79	O	0x6f	111	o
0x10	16	DLE (data link escape)	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1 (device control 1)	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2 (device control 2)	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3 (device control 3)	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4 (device control 4)	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK (negative ack)	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN (synchronous idle)	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB (end transmission)	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN (cancel)	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM (end of medium)	0x39	57	9	0x59	89	Y	0x79	121	y
0x1a	26	SUB (substitute)	0x3a	58	:	0x5a	90	Z	0x7a	122	z
0x1b	27	FSC (escape)	0x3b	59	;	0x5b	91	[	0x7b	123	{
0x1c	28	FS (file separator)	0x3c	60	<	0x5c	92	\	0x7c	124	|
0x1d	29	GS (group separator)	0x3d	61	=	0x5d	93	]	0x7d	125	}
0x1e	30	RS (record separator)	0x3e	62	>	0x5e	94	^	0x7e	126	~
0x1f	31	US (unit separator)	0x3f	63	?	0x5f	95	_	0x7f	127	DEL

Appendix: C Function Reference

This appendix provides a short summary of C library functions used in the code snippets. The descriptions are taken from "The C Library Reference Guide" by Eric Huss.

strcpy: `char *strcpy(char *str1, const char *str2)`
Copies the string pointed to by str2 to str1. Copies up to and including the null character of str2. If str1 and str2 overlap the behavior is undefined. Returns the argument str1.

strncpy: `char *strncpy(char *str1, const char *str2, size_t n)`
Copies up to n characters from the string pointed to by str2 to str1. Copying stops when n characters are copied or the terminating null character in str2 is reached. If the null character is reached, the null characters are continually copied to str1 until n characters have been copied. Returns the argument str1.

malloc: `void *malloc(size_t size)`
Allocates the requested memory and returns a pointer to it. The requested size is size bytes. The value of the space is indeterminate. On success a pointer to the requested space is returned. On failure a null pointer is returned.

realloc: `void *realloc(void *ptr, size_t size)`
Attempts to resize the memory block pointed to by ptr that was previously allocated with a call to malloc or calloc. The contents pointed to by ptr are unchanged. If the value of size is greater than the previous size of the block, then the additional bytes have an undeterminate value. If the value of size is less than the previous size of the block, then the difference of bytes at the end of the block are freed. On success a pointer to the memory block is returned (which may be in a different location as before). On failure or if size is zero, a null pointer is returned.

gets: `char *gets(char *str)`
Reads a line from stdin and stores it into the string pointed to by str. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first. The newline character is not copied to the string. A null character is appended to the end of the string. On success a pointer to the string is returned. On error a null pointer is returned. If the end-of-file occurs before any characters have been read, the string remains unchanged.

system: `int system(const char *string)`
The command specified by string is passed to the host environment to be executed by the command processor. A null pointer can be used to inquire whether or not the command processor exists. If string is a null pointer and the command processor exists, then zero is returned. All other return values are implementation-defined.

getenv: `char *getenv(const char *name)`
Searches for the environment string pointed to by name and returns the associated value to the string. This returned value should not be written to. If the string is found, then a pointer to the string's associated value is returned. If the string is not found, then a null pointer is returned.

execv: `int execv(const char *path, char *const argv[])`
Replaces the current process image with a new process image specified in path. The execv() function provide an array of pointers (argv) to null-terminated strings that represent the argument list available to the new program. The first argument should point to the filename associated with the file being executed. The array of pointers must be terminated by a null pointer.

Appendix: ASCII Table

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL (null)	0x20	32	space	0x40	64	@	0x60	96	‘
0x01	1	SOH (start of heading)	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX (start of text)	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX (end of text)	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT (end of transmission)	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ (enquiry)	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK (acknowledge)	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL (bell)	0x27	39	’	0x47	71	G	0x67	103	g
0x08	8	BS (backspace)	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB (horizontal tab)	0x29	41	)	0x49	73	I	0x69	105	i
0x0a	10	LF (new line)	0x2a	42	*	0x4a	74	J	0x6a	106	j
0x0b	11	VT (vertical tab)	0x2b	43	+	0x4b	75	K	0x6b	107	k
0x0c	12	FF (form feed)	0x2c	44	,	0x4c	76	L	0x6c	108	l
0x0d	13	CR (carriage return)	0x2d	45	-	0x4d	77	M	0x6d	109	m
0x0e	14	SO (shift out)	0x2e	46	.	0x4e	78	N	0x6e	110	n
0x0f	15	SI (shift in)	0x2f	47	/	0x4f	79	O	0x6f	111	o
0x10	16	DLE (data link escape)	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1 (device control 1)	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2 (device control 2)	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3 (device control 3)	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4 (device control 4)	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK (negative ack)	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN (synchronous idle)	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB (end transmission)	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN (cancel)	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM (end of medium)	0x39	57	9	0x59	89	Y	0x79	121	y
0x1a	26	SUB (substitute)	0x3a	58	:	0x5a	90	Z	0x7a	122	z
0x1b	27	FSC (escape)	0x3b	59	;	0x5b	91	[	0x7b	123	{
0x1c	28	FS (file separator)	0x3c	60	<	0x5c	92	\	0x7c	124	|
0x1d	29	GS (group separator)	0x3d	61	=	0x5d	93	]	0x7d	125	}
0x1e	30	RS (record separator)	0x3e	62	>	0x5e	94	^	0x7e	126	~
0x1f	31	US (unit separator)	0x3f	63	?	0x5f	95	_	0x7f	127	DEL

Appendix: 64-bit Linux Syscall List

Nr.	Name	RAX	RDI	RSI	RDX	R10	R8
0	sys_read	0x00	unsigned int fd	char *buf	size_t count	-	-
1	sys_write	0x01	unsigned int fd	const char *buf	size_t count	-	-
2	sys_open	0x02	const char *filename	int flags	int mode	-	-
3	sys_close	0x03	unsigned int fd	-	-	-	-
10	sys_mprotect	0x0a	unsigned long start	size_t len	unsigned long prot	-	-
40	send_file	0x28	int out_fd	int in_fd	off_t *offset, size_t count		
59	sys_execve	0x3b	const char *filename	const char **argv	const char **envp	-	-
60	sys_exit	0x3c	int exit_code	-	-	-	-