SASD
Winter Term 2018
Exam 1
25.01.2019
Time Limit: 90 Minutes

Name: ______________________________

Immatriculation Number: ______________________________

This exam contains 9 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.

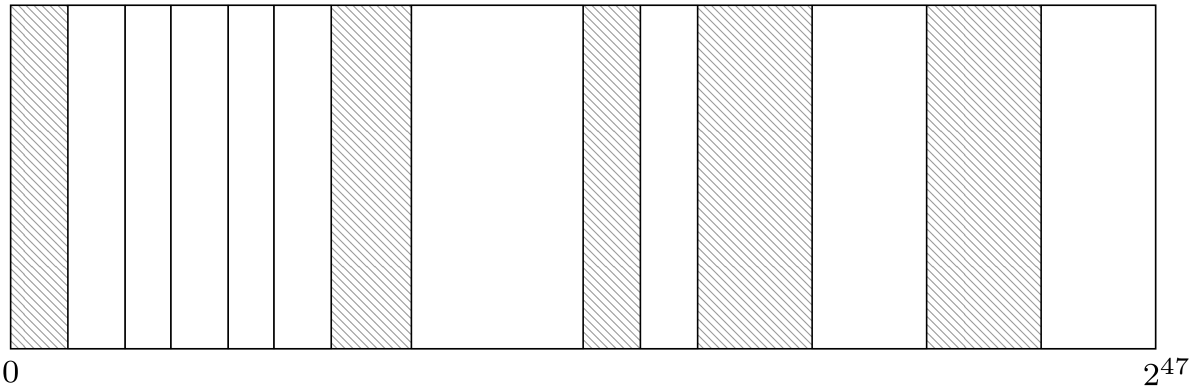
You may *not* use your books, notes, or any calculator on this exam.

- 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	4	
Total:	44	

1. (10 points) **Low Level**



(a) (4 points) Name the 9 sections of a typical x86-64 user-space application on a recent Linux in the above figure.

(b) (1 point) Which sections are writable?

(c) (1 point) Which sections are executable?

(d) (1 point) Which sections are both writable and executable?

(e) (3 points) In which section does the following snippet occur most likely?

```
14c0:  ff 35 72 79 20 00  pushq  0x207972(%rip)
14c6:  ff 25 74 79 20 00  jmpq   *0x207974(%rip)
14cc:  0f 1f 40 00        nopl   0x0(%rax)
```

2. (10 points) **Memory Corruption**

```c
int main(int argc, char* argv[]) {
  if(argc != 2) return -1;

  char* password = argv[1];
  unsigned char password_length = strlen(password);

  if(password_length >= 5 && password_length < 12) {
    if(strncmp(password + 12, "SASD", 4) == 0) {
      system("/bin/bash");
    }
  } else {
    puts("Wrong password length\n");
  }
  return 0;
}
```

(a) (2 points) Describe all memory safety violations which you find in the program.

(b) (4 points) Give a sample input which opens a shell and explain what it does.

(c) (2 points) Why is the length check circumventable?

(d) (2 points) How can you fix the program?

3. (10 points) **Defensive Programming**

   Let's assume we use a buggy password manager. It has all kinds of flaws, including use-after-free bugs, format string vulnerabilities, and potential buffer overflows on the stack.

   (a) (2 points) Briefly explain the advantages that sandboxing provides in this scenario.

   (b) (2 points) What are the limitations of sandboxing in this scenario?

   (c) (3 points) Can the attacker still mount an attack? If so, describe which attack and how it is mounted.

   (d) (3 points) Briefly describe an attack which is perfectly mitigated by using virtualization or full system emulation but not mitigated by using a sandbox.

4. (10 points) **Exploits**

You control the stack of a vulnerable program, which uses no libc and has non-executable buffers. Given is a part of the memory contents. Construct a ROP chain on the stack frame by filling in values/addresses.

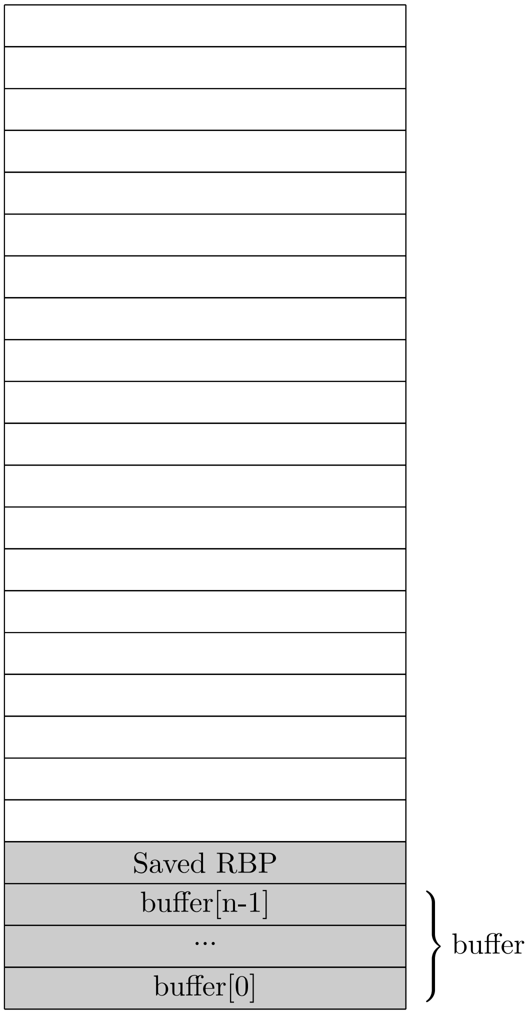
Your ROP chain should add the line "`u:x:1002:1002:,,,:/home/u:/bin/sh`" to the file "`/etc/passwd`" when the current function returns.

Hints:

- A syscall return value is in `RAX`.
- The file mode is ignored (can be 0).
- The flag for write and append (`O_RDWR | O_APPEND`) is 1026.
- You do not have to close the file.

ASM	Hex
`pop RAX; ret`	`58 C3`
`pop RBX; ret`	`5B C3`
`pop RCX; ret`	`59 C3`
`pop RDX; ret`	`5A C3`
`pop RSI; ret`	`5E C3`
`pop RDI; ret`	`5F C3`
`xchg RAX, RDI; ret`	`97 C3`
`inc RAX; ret`	`48 FF C0 C3`
`xor RAX, RAX; ret`	`48 31 C0 C3`
`syscall; ret`	`0F 05 C3`

Gadget Cheat Sheet



Current Stack Frame

```
Address     0  1  2  3   4  5  6  7   8  9  A  B   C  D  E  F   0123 4567 89AB CDEF
-------------------------------------------------------------------------------------
01745000   54 0c 9e 30  a5 20 9b 36  93 a5 5f bb  d8 55 5a b9  |T..0|. .6|.._.|.UZ.|
01745010   c9 33 2f 5c  31 59 c3 a6  4b 05 0f 05  c3 c9 b7 ba  |.3/\|1Y..|K...|....|
01745020   91 0f 61 cf  f9 e3 ee ee  d7 15 61 f4  b8 d3 97 c3  |..a.|....|..a.|....|
01745030   f0 6d be e1  42 17 5a c3  a4 ea 2f 65  74 63 2f 70  |.m..|B.Z.|../e|tc/p|
01745040   61 73 73 77  64 00 a8 37  ab 6c 56 00  f0 ea 58 c3  |assw|d..7|.lV.|..X.|
01745050   13 e6 f4 ec  5b c3 61 a8  b9 78 e9 0c  b4 3f 80 d9  |....|[.a.|.x..|.?..|
01745060   d7 43 83 ba  c4 5e c3 dd  dd 0f d7 f1  d5 b6 de b6  |.C..|.^..|....|....|
01745070   26 cb 6d 75  3a 78 3a 31  30 30 32 3a  31 30 30 32  |&.mu|:x:1|002:|1002|
01745080   3a 2c 2c 2c  3a 2f 68 6f  6d 65 2f 75  3a 2f 62 69  |:,,,|:/ho|me/u|:/bi|
01745090   6e 2f 73 68  00 d6 08 8f  df 04 d4 e7  99 5f c3 e6  |n/sh|....|....|._..|
-------------------------------------------------------------------------------------
```

Memory Dump from `0x01745000` *to* `0x0174509f`

5. (4 points) **(Bonus) Lecture Challenges**

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

Lecture challenge username: ______________________

(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 (format)*
What was the limitation in the format string attack?

(d) (0.5 points) *Challenge #4 (needle)*
Which git command did you use to solve the challenge?

(e) (0.5 points) *Challenge #5 (mystery)*
What was the target architecture of the mysterious binary?

(f) (0.5 points) *Challenge #6 (shellcode)*
Which helper tool(s) did you use to write the shellcode?

(g) (0.5 points) *Challenge #7 (secwrap)*
Which function is used to apply all the seccomp rules?

(h) (0.5 points) *Challenge #8 (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	ˆ	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: 32-bit Linux Syscall List

Nr.	Name	EAX	EBX	ECX	EDX	ESI	EDI
1	sys_exit	0x01	int exit_code	-	-	-	-
2	sys_fork	0x02	-	-	-	-	-
3	sys_read	0x03	unsigned int fd	char *buf	size_t count	-	-
4	sys_write	0x04	unsigned int fd	const char *buf	size_t count	-	-
5	sys_open	0x05	const char *filename	int flags	int mode	-	-
6	sys_close	0x06	unsigned int fd	-	-	-	-
7	sys_waitpid	0x07	pid_t pid	int *stat_addr	int options	-	-
8	sys_creat	0x08	const char *pathname	int mode	-	-	-
9	sys_link	0x09	const char *oldname	const char *newname	-	-	-
10	sys_unlink	0x0a	const char *pathname	-	-	-	-
11	sys_execve	0x0b	const char *filename	const char **argv	const char **envp	-	-
12	sys_chdir	0x0c	const char *filename	-	-	-	-
13	sys_time	0x0d	time_t *tloc	-	-	-	-
14	sys_mknod	0x0e	const char *filename	int mode	unsigned dev	-	-
15	sys_chmod	0x0f	const char *filename	mode_t mode	-	-	-
16	sys_lchown16	0x10	const char *filename	old_uid_t user	old_gid_t group	-	-
19	sys_lseek	0x13	unsigned int fd	off_t offset	unsigned int origin	-	-
20	sys_getpid	0x14	-	-	-	-	-
26	sys_ptrace	0x1a	long request	long pid	long addr	long data	-
37	sys_kill	0x25	int pid	int sig	-	-	-
88	sys_reboot	0x58	int magic1	int magic2	unsigned int cmd	void *arg	-
125	sys_mprotect	0x7d	unsigned long start	size_t len	unsigned long prot	-	-

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	-	-
59	sys_execve	0x3b	const char *filename	const char **argv	const char **envp	-	-
60	sys_exit	0x3c	int exit_code	-	-	-	-