

**SASD**  
**Winter Term 2017**  
**Exam 1**  
**06.12.2017**  
**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**

At the beginning of this month, a new CPU vulnerability with the name “Meltdown” was released. Meltdown allows to read kernel memory from an unprivileged user space application.

(a) (2 points) Which mechanism should prevent an application from reading kernel memory?

(b) (4 points) Sketch the full virtual address space (user + kernel) of an application on a typical 64-bit system, including the 8 user space regions discussed in the lecture

(c) (2 points) Give the valid address range (in hexadecimal or powers of 2) for

(a) the user space

(b) the kernel space

(d) (2 points) Meltdown accesses an invalid address, which causes a segmentation fault. Describe a possibility to catch segmentation faults in C programs.

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

(a) (2 points) Name the two types of memory safety violations

(b) (4 points) For each of the two types name two examples

(c) (4 points) For each of your 4 examples, provide a short code snippet demonstrating it

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

A fuzzer was able to crash the following code snippet:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

void upperCaseTags(char* string) {
    int i;
    for(i = 0; i < strlen(string); i++) {
        if(string[i] == '<') {
            while(string[++i] != '>') {
                string[i] = toupper(string[i]);
            }
        }
    }
}

int main(int argc, char* argv[]) {
    if(argc > 1) {
        size_t len = 0;
        FILE* f = fopen(argv[1], "r");
        char* buffer = NULL;
        if(f) {
            getline(&buffer, &len, f);
            upperCaseTags(buffer);
            printf("%s\n", buffer);
            free(buffer);
            fclose(f);
        }
    }
}
```

The fuzzer found that the following input lead to a crash: h<>A<>d<SC<1><o

However, running this without the fuzzer does not crash and the output is: h<>A<>d<SC<L><O

(a) (2 points) What is the function supposed to do?

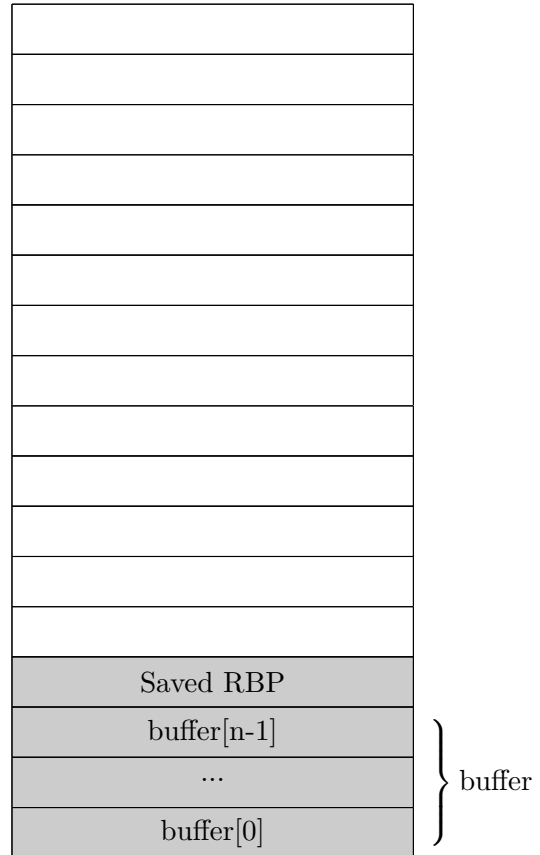
- (b) (2 points) What is the bug in the program?
- (c) (2 points) Give an input which is as short as possible that also crashes the program.
- (d) (2 points) Describe a tool/technique and how to use it to figure out why this input crashes sometimes.
- (e) (2 points) Provide a fix for the buggy function.

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 print “SASD2017” to the standard output when the current function returns.

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
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
17105000	5a	c3	00	00	e8	5c	fe	ff	ff	5b	c3	c6	48	8b	55	a8	Z...	\...	[...	H.U.
17105010	48	8d	45	b0	48	89	d1	48	89	f2	48	8d	35	ed	00	00	H.E.	H..H	..H	5...
17105020	00	48	58	c3	b8	00	00	00	00	e8	57	fe	2f	62	69	6e	.HX.	....	..W	/bin
17105030	2f	73	68	00	c7	e8	33	fe	ff	ff	90	48	8b	45	f8	64	/sh.	..3.	...H	E.d
17105040	48	33	04	25	59	c3	00	00	74	05	e8	5f	c3	ff	ff	c9	H3.%	Y...	t...	....
17105050	c3	55	48	89	e5	48	83	ec	10	89	7d	fc	48	89	75	0f	.UH.	..H...	}.	H.u.
17105060	05	c3	fc	01	7e	13	48	8b	45	f0	48	83	c0	08	48	8b	...	~.H.	E.H.	..H.
17105070	5e	c3	89	c7	e8	4c	ff	ff	ff	b8	00	00	00	00	c9	c3	^...	L...	....	....
17105080	0f	1f	44	00	00	41	57	41	56	41	89	53	41	53	44	32	.D.	AWA	VA.S	ASD2
17105090	30	31	37	00	00	20	00	55	48	8d	2d	26	05	20	00	53	017.	..U	H.-&	l. .S

*Memory Dump from 0x17105000 to 0x1710509f*

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

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

**Fuzzy Land username:** \_\_\_\_\_

(a) (0.5 points) *Challenge #1 (minielf)*

What identifies a file as an ELF binary?

(b) (0.5 points) *Challenge #2 (quadfloat)*

What are the three parts of a IEEE 754 quadruple-precision binary floating-point number?

(c) (0.5 points) *Challenge #3 (format)*

How can a printf parameter be directly addressed?

(d) (0.5 points) *Challenge #4 (needle)*

Which arguments does git bisect require?

(e) (0.5 points) *Challenge #5 (mystery)*

How many bits does the target architecture of the mysterious binary have?

(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 enables 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: 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	RBX	RCX	RDX	RSI	RDI
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	-	-	-	-