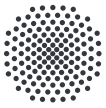

University of Stuttgart
Institute of
Information Security

System and Web Security
Summer Term 2018
Prof. Dr. Ralf Küsters

Homework 2

Submission via ILIAS until May 02, 2018, 07:59 (One day later than usual because of the holiday.).

General Notes

- If you encounter difficulties, you SHOULD[1] ask the teaching assistants in the consultation hour.
- To solve the homework, you SHOULD form teams of 3 people.
- Your team size MUST NOT exceed 3 people.
- You MUST submit your homework in ILIAS. (In the ILIAS course, you can find the *Homework Submissions* item. There, you can register your team and submit your solution.)
- You are free to choose whether you write your solutions in German or in English.
- Your submission MUST include all team member's names and matriculation numbers.
- You MUST use the System Security Virtual Machine to work on all problems (except if stated otherwise).
- You MUST submit your solution as a PDF or TXT document. If your solution contains source code, you MUST submit your source code in its native file format separately. In your main document, it is sufficient to point to these source code files.
- When you carry out attacks, you SHOULD take a screenshot to document that your exploit worked (and include this screenshot in your submission).
- Your team gains one point for each problem if solved reasonably.
- If you do not adhere to these rules, you risk loosing points.

Information for Solving the Problems

- Many problems of the homework depend on the computer architecture you use. In ILIAS, you can download the *System Security Virtual Machine (VM)*. This VM serves as our reference system. Other systems may (or most likely will) produce different results that we cannot grade as correct. **You MUST use this VM to work on all problems (except if stated otherwise).**
- Please note that you can find further information on how to install the VM in ILIAS.
- When you compile source code, you MUST use `gcc` with the following options:

  -fno-stack-protector disables several defence mechanisms

  -O0 disables optimizations

  -ggdb3 adds meta information for the debugger to the binary

  Although we have configured the shell of the VM such that the `gcc` command uses these options automatically, you should not omit these (especially when writing your own make files, etc.). For example, you compile a C program (contained in file `myprog.c`) like this:

  ```
  gcc -fno-stack-protector -O0 -ggdb3 -o myprog myprog.c
  ```

[1] SHOULD, MUST, and MUST NOT are used as defined in RFC2119.

Problem 1: Buffer Overflow

Show and explain the output of the C program below.

File: code/sof–simple–example2.c

```c
#include <stdio.h>
#include <string.h>
char foo[] = "zweiundvier\0zig";
void function () {
        char buffer1[8];
        char buffer2[3];
        char buffer3[4];
        strcpy(buffer3, foo);
        printf("%s\n", buffer2);
}
int main () {
        function();
}
```

Problem 2: Stack-based Overflow

The C program below takes one argument. This argument is supposed to be a filename. The program opens that file and reads 512 bytes from it.

Provide a file which contains an exploit string that forces the program to start a shell. Add a screenshot to your submission that shows that your attack is successful. Why does your attack work?

Hint: You can create such a file with, for example, a hex editor or an other C program that generates such a file. You can use the shell code from the example files provided with the lecture.

File: code/sof–exploit–7–file–target.c

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

void callme(char *filename) {
  char b[256];
  FILE* f;

  f = fopen(filename, "r");
  fgets(b, 512, f);
}

int main(int argc, char *argv[]) {
  if (argc > 1)
    callme(argv[1]);
  return 0;
}
```

Problem 3: Stack-based Overflow

Provide a C program which executes the C program below and forces this program to start a shell. Add a screenshot to your submission that shows that your attack is successful. Why does your attack work?

File: code/sof–exploit–8–target.c

```c
#include <stdio.h>
#include <string.h>
void foo (char *str) {
  int i;
  char str1[256];
  char str2[512];
  strcpy(str2,str);
  for (i = 0; i < 256; i++)
    str1[i] = str2[2*i];
  str1[255] = '\0';
  printf("%s\n", str1);
}
void test(char *x){
  if (!strncmp(x,"login=",6)) {
    foo(x);
  }
}
main(int argc, char *argv[]) {
  if (argc > 1) {
    test(argv[1]);
  }
}
```

Problem 4: Memory Alignment

Inform yourself about *memory alignment*. How are the variables `a` to `k` aligned on the stack in the C program below? Which pattern regarding the memory addresses do you observe?

File: code/c–alignment–2.c

```c
void do_something() {
  int a;
  int b;
  char c;
  int d;
  short e;
  int f;
  char g;
  short h;
  int i;
  char j[3];
  int k;

  a = 1;
  b = 2;
  d = a+b;
}
```

```
18
19 int main(){
20   do_something();
21 }
```