TECHNISCHE UNIVERSITÄT DRESDEN

**Department of Computer Science**, Institute of System Architecture, Operating Systems Group

# Distributed Operating Systems

## Exercise 3: Buffer Overflows

In the tutorial, all solutions will be presented by students. Please be prepared for all questions as the exercise will focus on discussion, not on understanding the question and gathering the knowledge.

---

### Working with GNU binutils

Download the sample binary and use GNU binutils to solve the following tasks:

1) The binary contains a function named `grant_rights()`. Use the `nm` tool to determine the function's address.
2) Use the `objdump` utility to disassemble the binary. Find the `main()` function and determine, which functions are called by `main()`.
3) Use `strace` to find out if (and with which arguments) the write system call is called.
4) Make yourself comfortable with the `gdb` debugger. Run the sample binary using `gdb` and:
   a) Single-step through the `do_work()` function.
   b) Determine the return address stored in `do_work()`'s stack frame.
   c) The binary uses `fread()` to read data into a buffer within the function `do_work()`. Using gdb, determine the address and the maximum size of the buffer.
5) Write a C program that takes 32-bit hexadecimal numbers from the command line and prints them byte-wise to `stdout`. Use `hexdump` to verify the results.

---

### Redirecting the Control Flow

6) Use the knowledge obtained in Tasks 1–5) to modify the `do_work()` function's return address so that it returns to the function `grant_rights()`.

## Executing Shell Code

7) Download the 32-byte shell code binary. Use `ndisasm` to disassemble the shell code and find out, what it does. Place it in the sample binary's buffer and overflow the buffer, so that the `do_work()` function returns into the shell code.

## Return to libC

8) Let's assume the existence of non-executable stacks. The method from Task 7) doesn't work under these circumstances. However, return-into-libC attacks still work. Overflow the buffer in a way that makes `do_work()` return into the libC's `execve()` function and uses this function to start `/bin/date`.

## Safe String Functions?

The following program `sn.c` passes its first command line argument to the function `do_work()`, which then uses `strncpy()` to make sure that the buffer `buf` cannot be overflown.

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

int do_work(char *str)
{
    char buf[32];
    strncpy(buf, str, 32);
    return strlen(buf);
}

int main(int argc, char **argv)
{
    if (argc != 2)
        return -1;
    printf("ret: %d\n", do_work(argv[1]));
    return 0;
}
```

9) Compile the program in the following ways. How predictable is the output of the program in both cases? Why?

```
$> gcc -fno-stack-protector -o sn_noprot sn.c
$> gcc -fstack-protector -o sn_prot sn.c
```