**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Презентация по английскому №3

«**Reverse Engineering**»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выполнил | |  | Принял | |
| ФИО: | Цыпышев Т. А. \_\_ |  | ФИО: | Юдина М. В.\_\_\_\_\_\_\_\_\_\_ |
| Группа: | ИУ5-11Б\_\_\_\_\_\_\_\_\_\_\_\_ |  | Должность: | Старший преподаватель |
| Дата: | 29.11.2022\_\_\_\_\_\_\_\_\_\_ |  | Дата: | 29.11.2022\_\_\_\_\_\_\_\_\_\_\_\_ |
| Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва, 2022 г.

**Оглавление презентации:**

• Reverse Engineering

• Introduction

• Introduction to the Architecture of Linux

• Learn more about the components

• Kernel Source Code

• Binary Disassembly

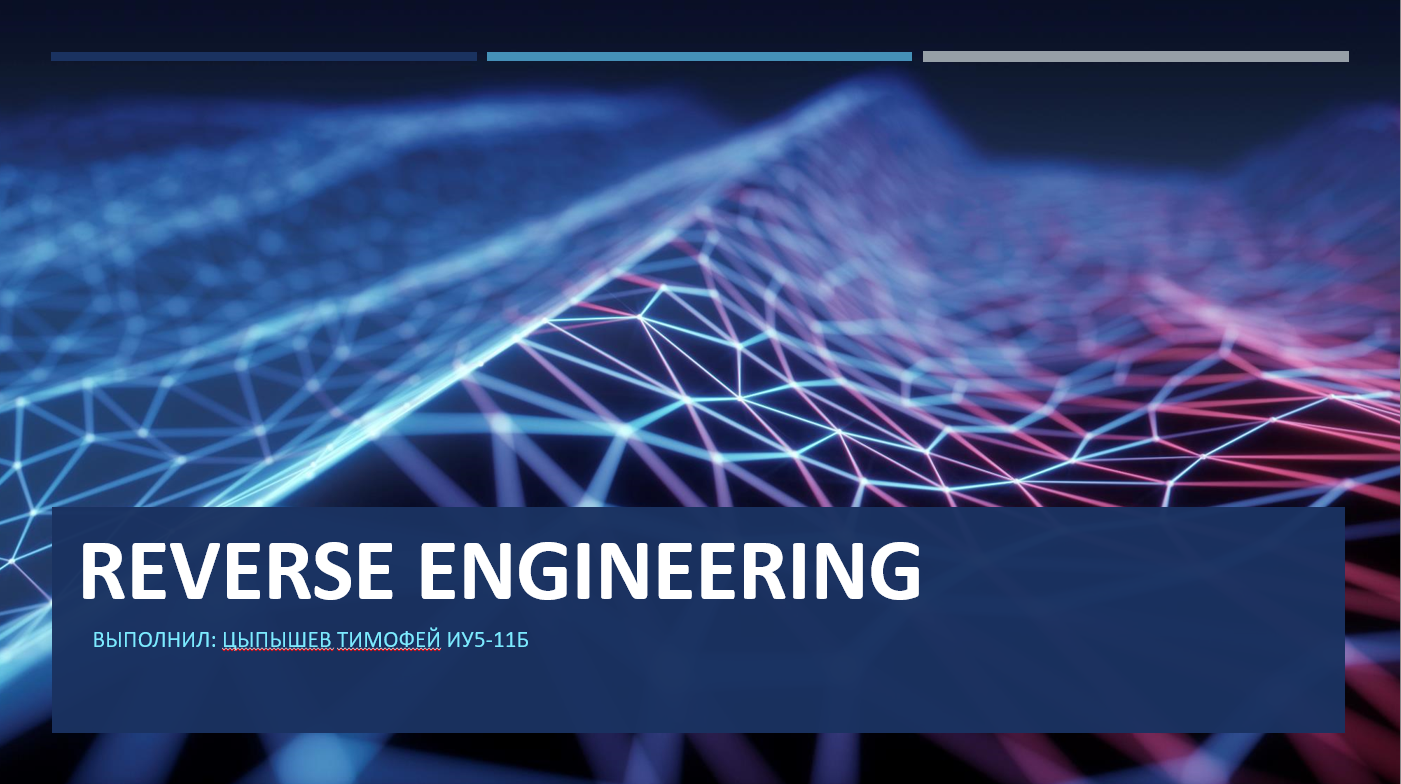
• A little bit about disassemblers

• Example of User Space Binary Disassembly

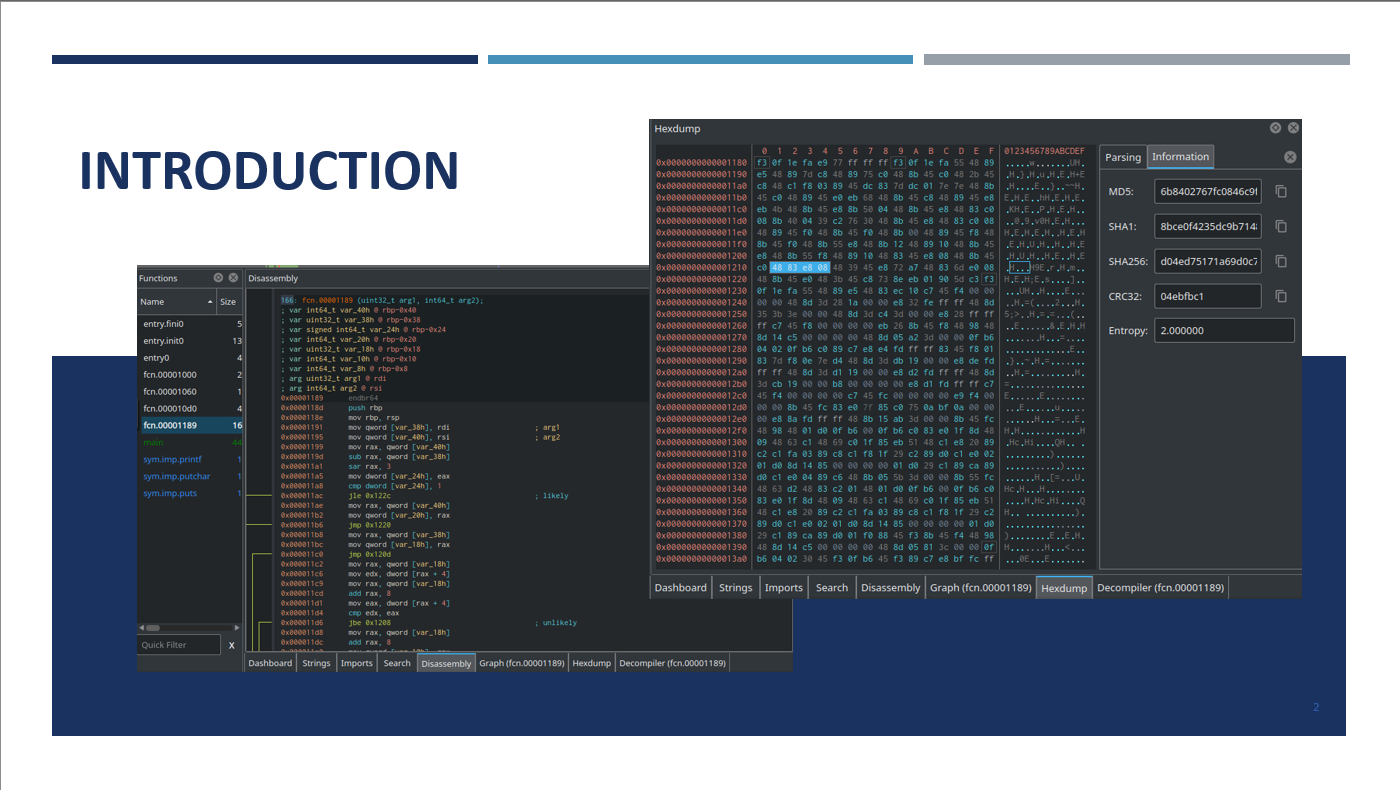
• Questions

• Thanks for watching

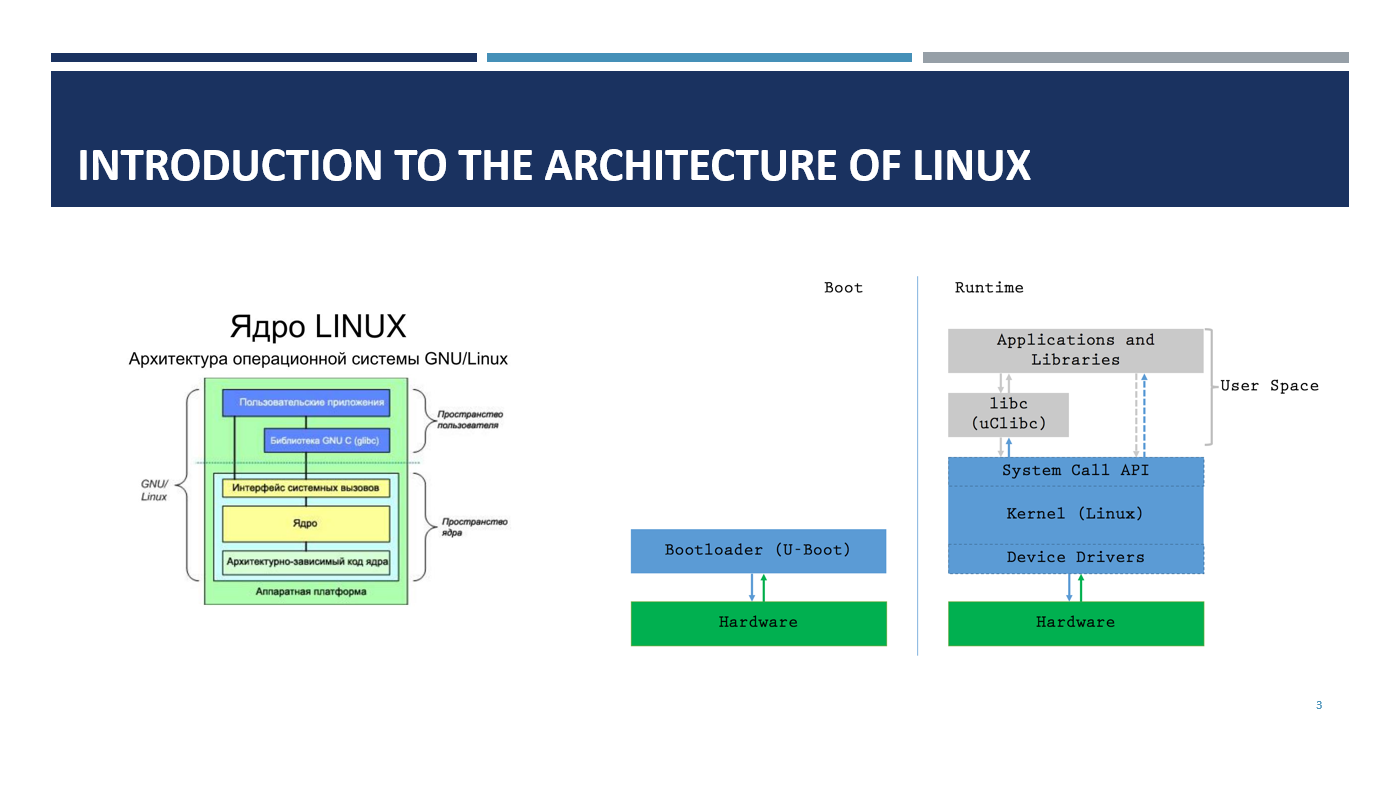
**1 слайд**



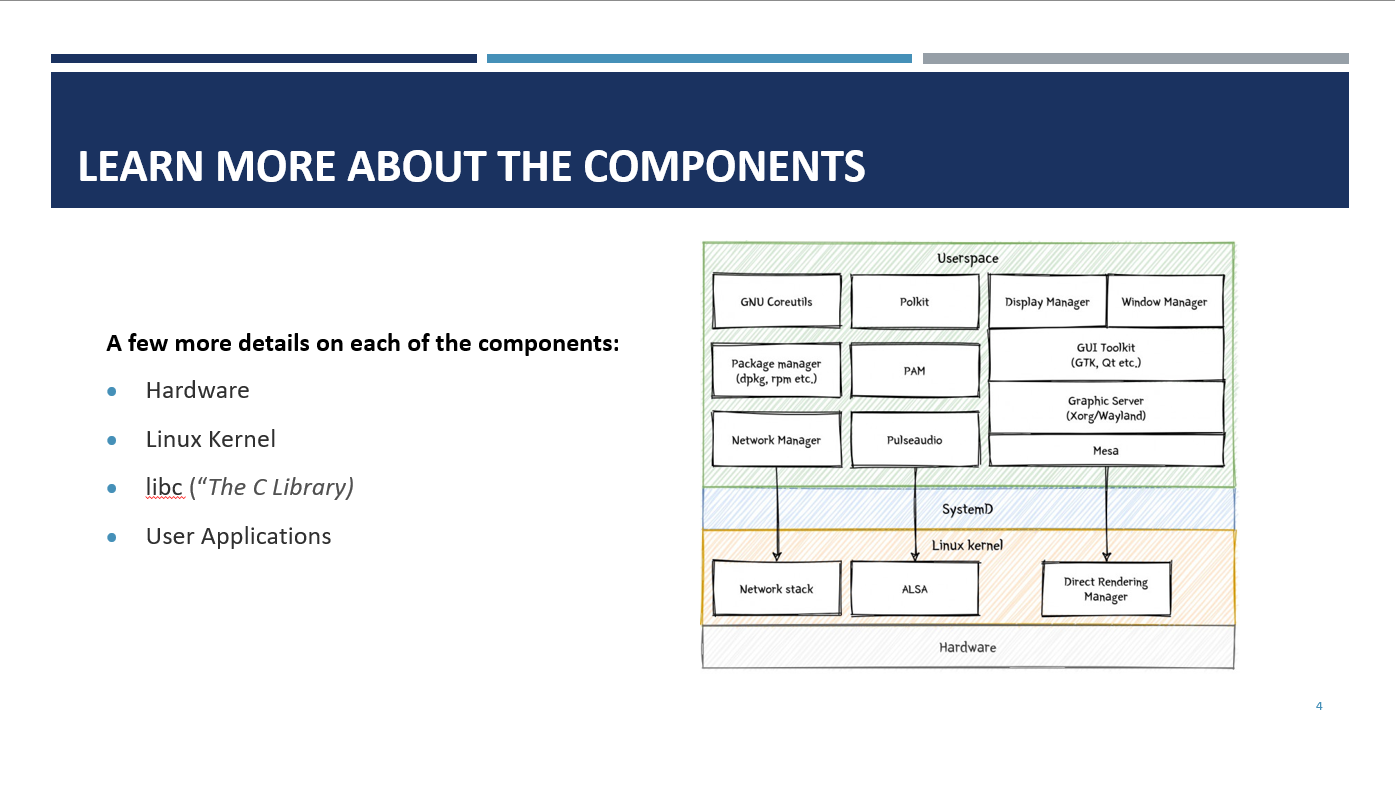
**2 слайд**



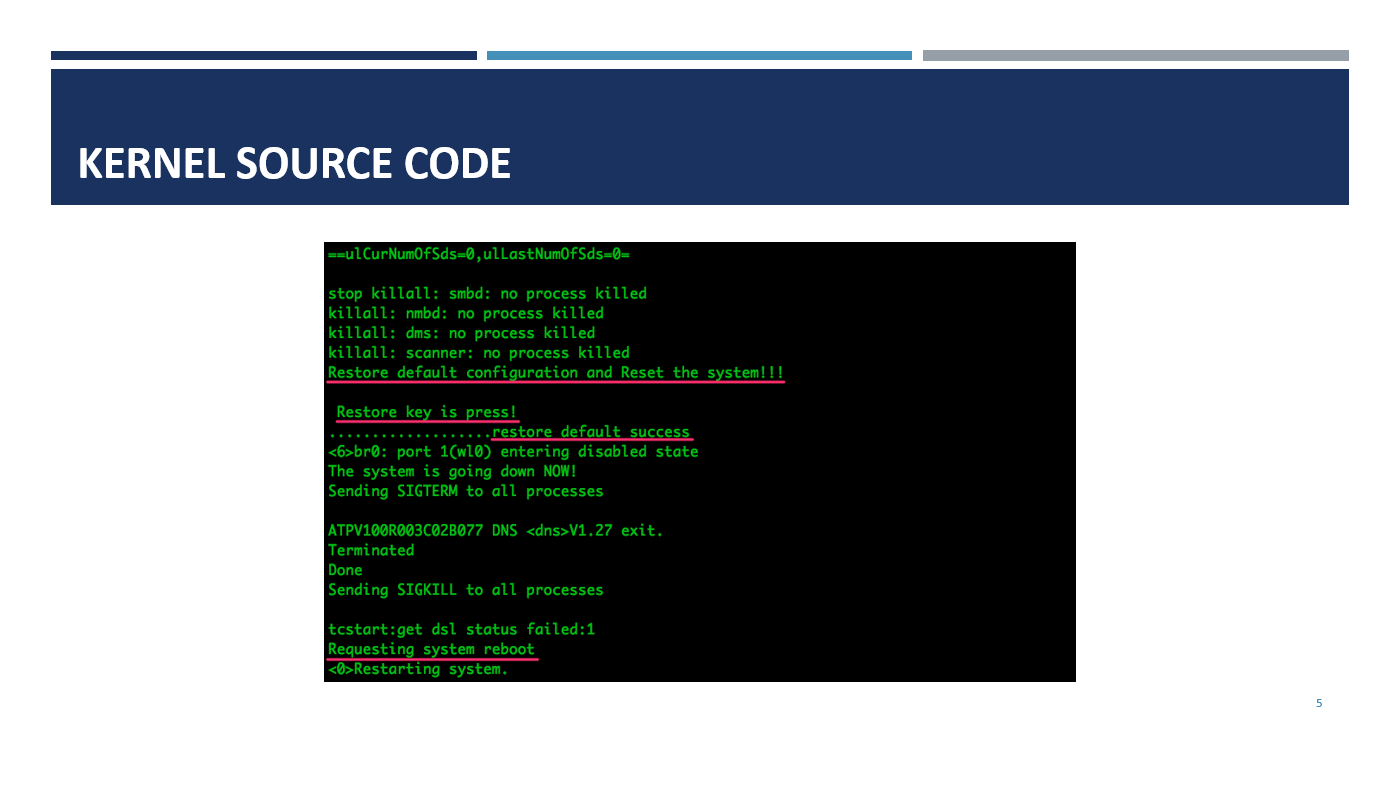
**3 слайд**



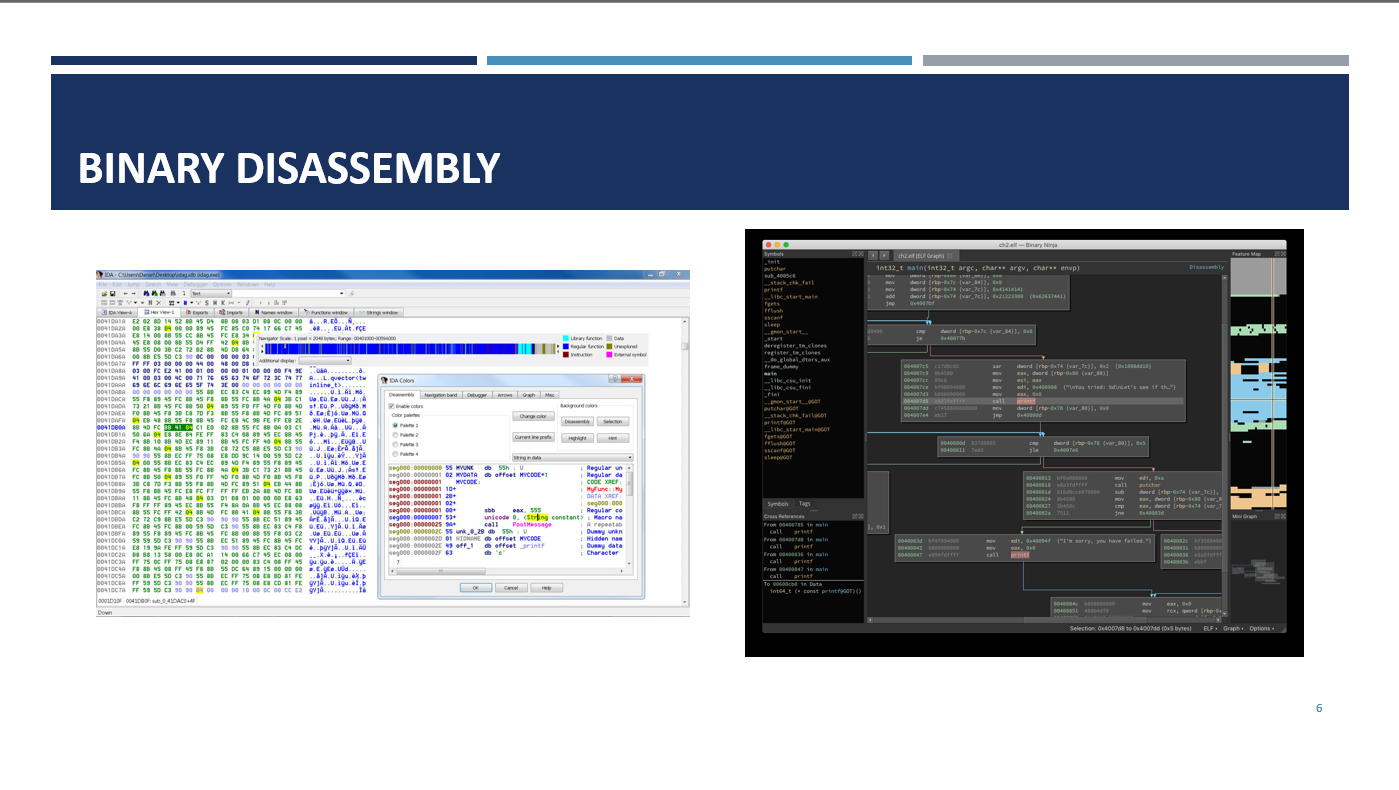
**4 слайд**



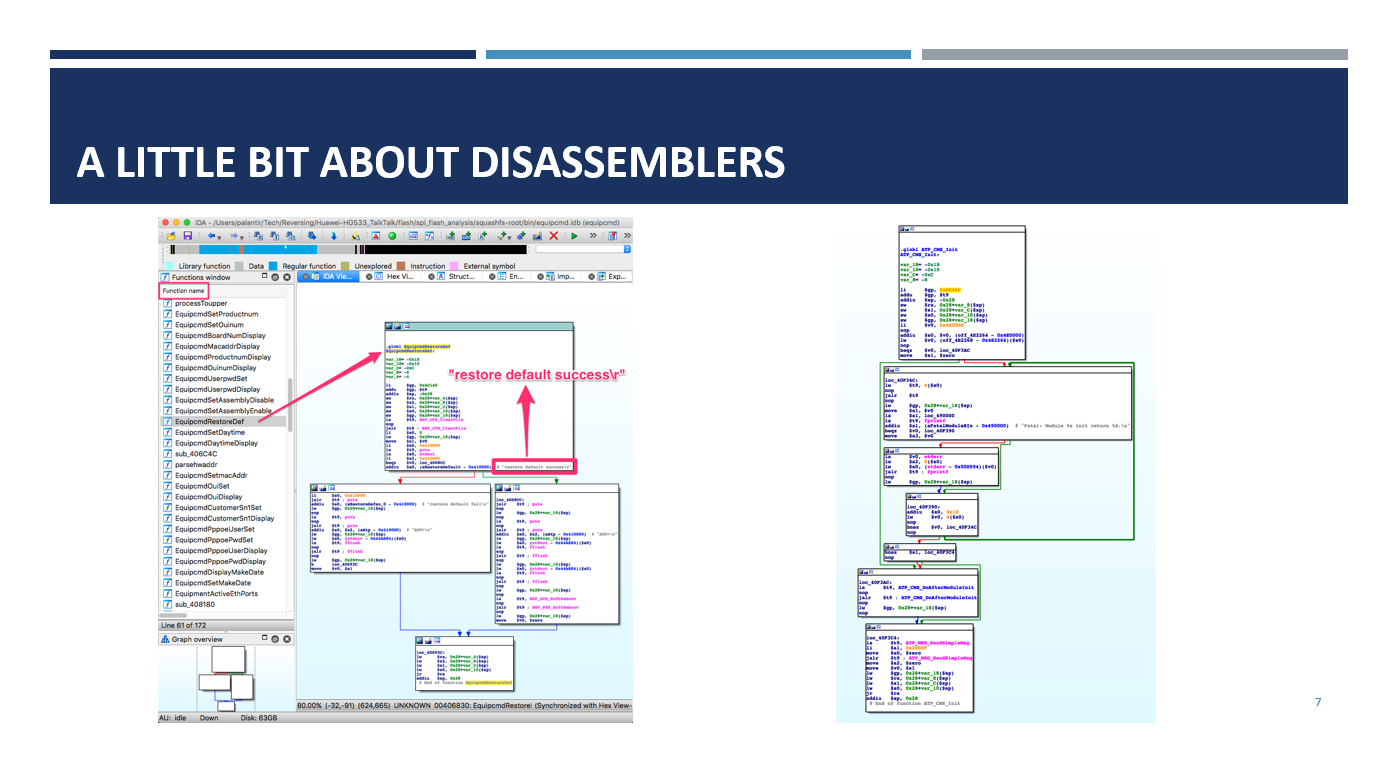
**5 слайд**



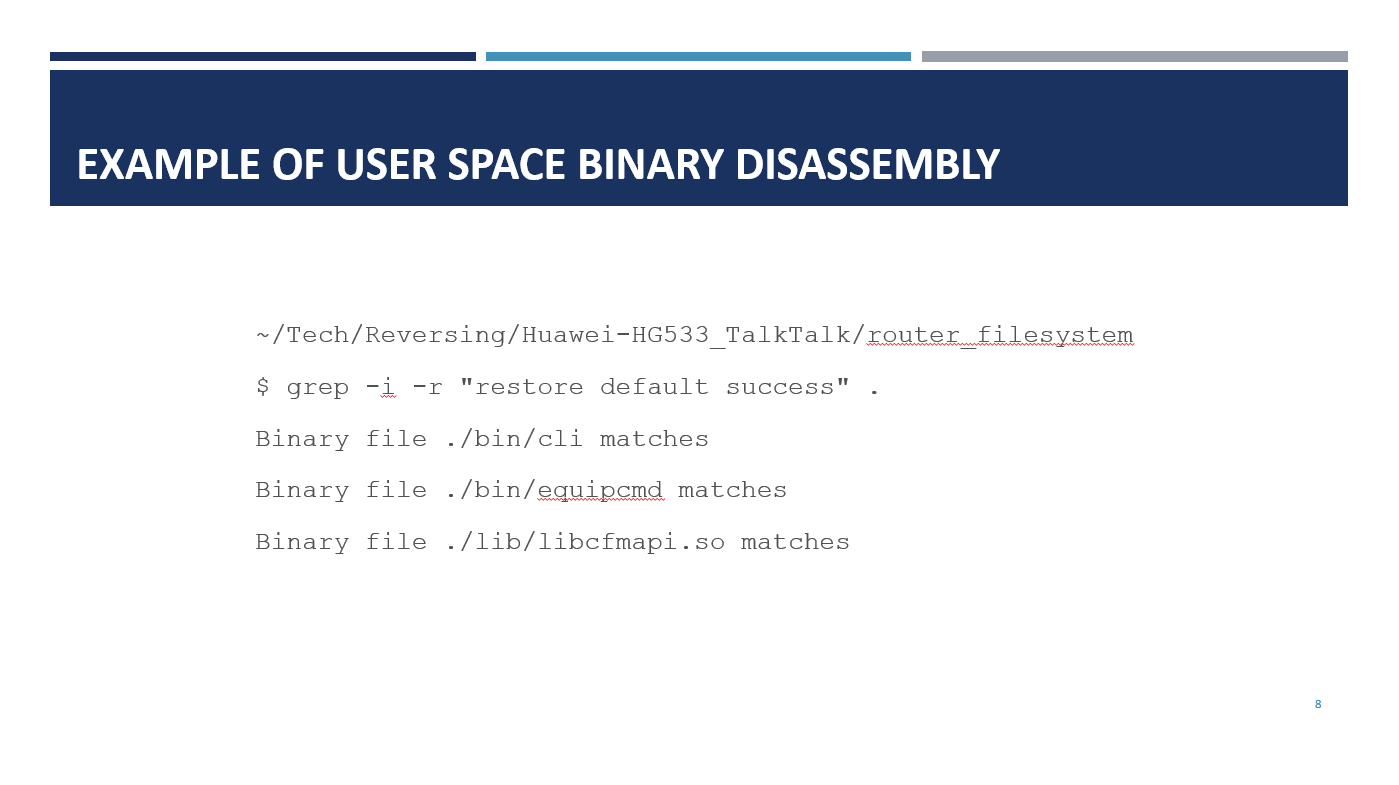
**6 слайд**



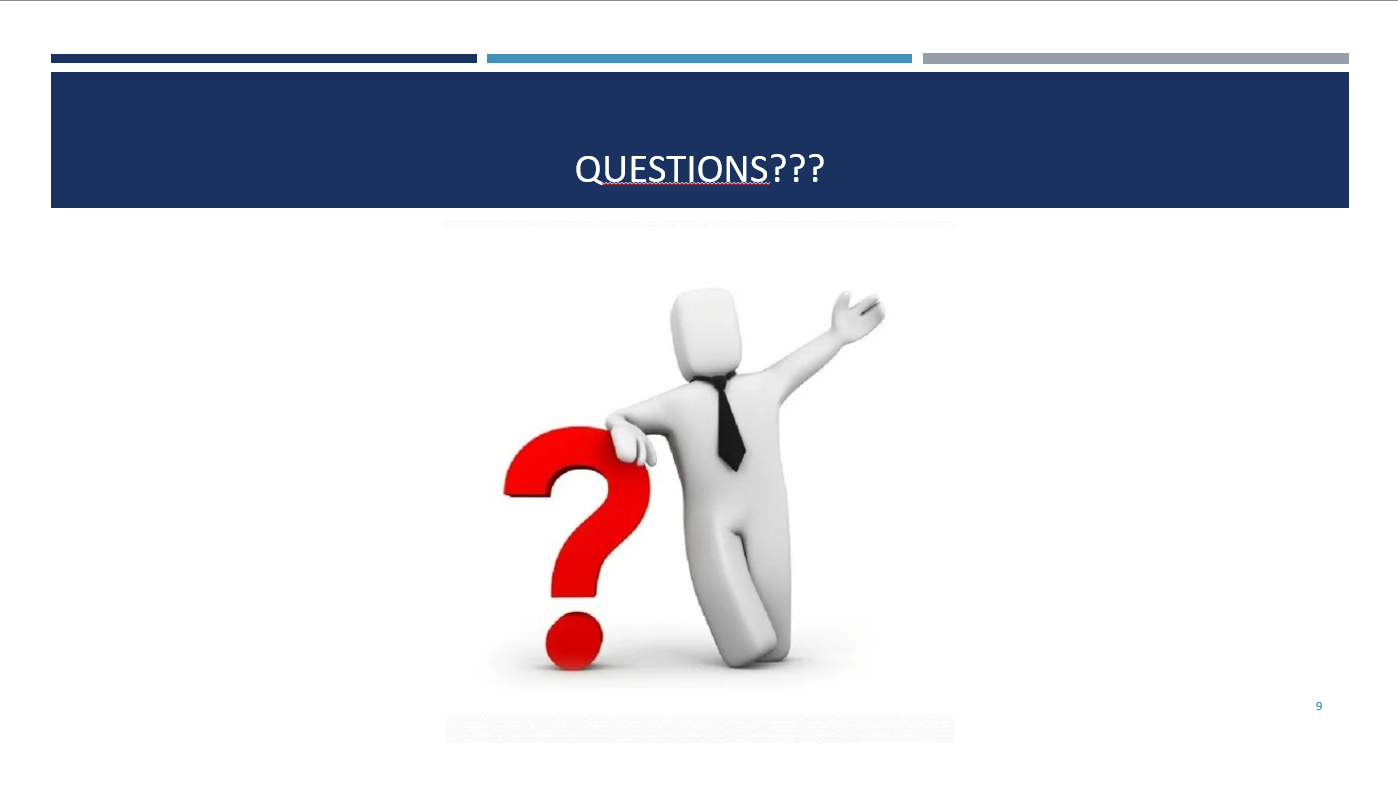
**7 слайд**



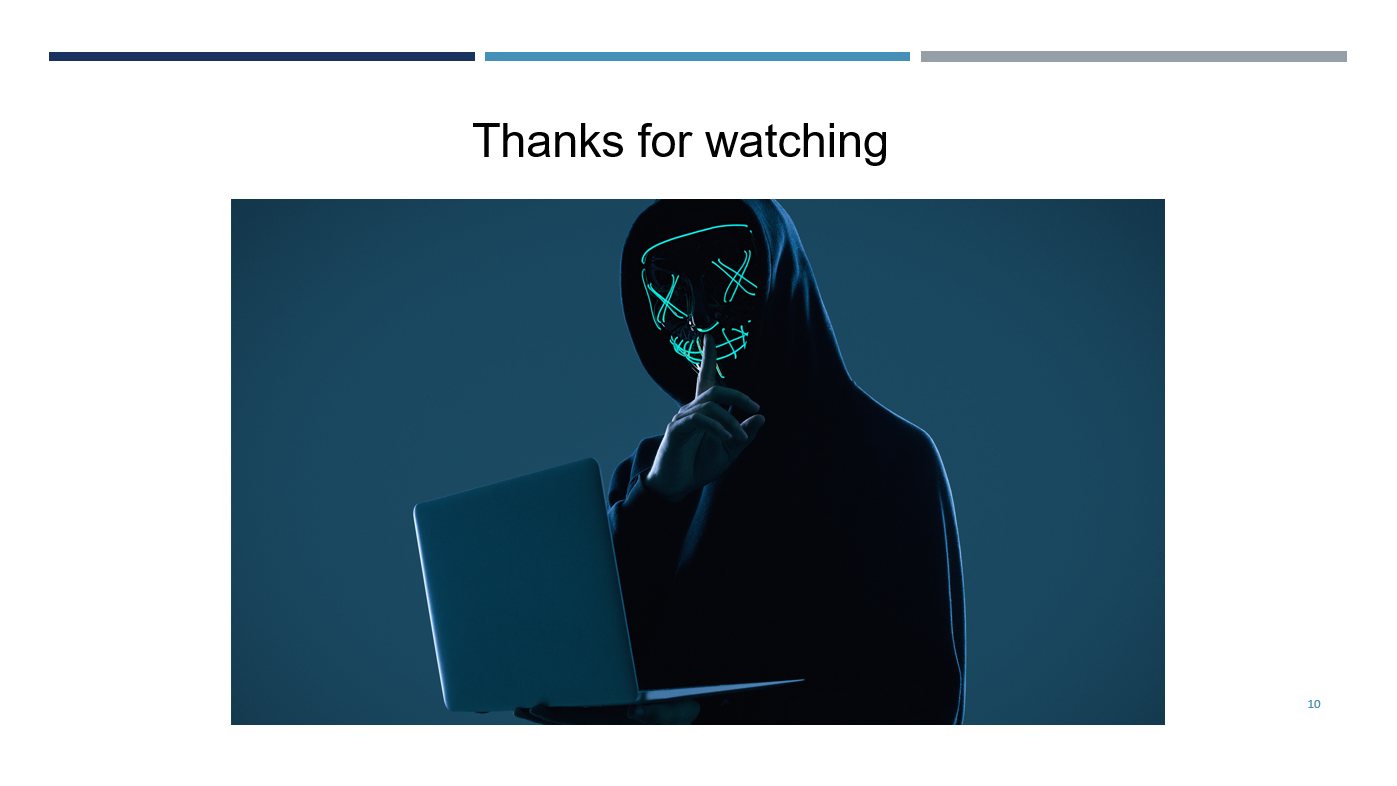
**8 слайд**



**9 слайд**



**10 слайд**



1. **Reverse Engineering**
2. **Introduction**

In this presentation, I'm going to explain some basic theory of Linux architecture, binary file disassembly, and other related concepts.

Suppose we have already extracted all the firmware from the router and unpacked it.

What should we do next?

1. **Introduction to the Architecture of Linux**

In order to know where to look for what, it’s important to understand the overall architecture of the system. Let’s quickly review this device’s:

The bootloader is the first piece of code to be executed on boot. Its job is to prepare the kernel for execution, jump into it and stop running. From that point on, the kernel controls the hardware and uses it to run user space logic.

1. **Learn more about the components**

A few more details on each of the components:

* **Hardware**: The CPU, Flash, RAM and other components are all physically connected
* **Linux Kernel**: It knows how to control the hardware. The developers take the Open Source Linux kernel, write *drivers* for their specific device and compile everything into an executable Kernel. It manages memory, reads and writes hardware registers, etc. In more complex systems, “kernel modules” provide the possibility of keeping device drivers as separate entities in the file system, and dynamically load them when required
* **libc** (“*The C Library*”): It serves as a general purpose wrapper for the System Call API, including extremely common functions like printf, malloc or system. Developers are free to call the system call API directly, but in most cases, it’s MUCH more convenient to use libc. Instead of the extremely common glibc (GNU C library) we usually find in more powerful systems, this device uses a version optimised for embedded devices: [uClibc](https://www.uclibc.org/).
* **User Applications**: Executable binaries in /bin/ and *shared objects* /lib/ (libraries that contain functions used by multiple binaries) comprise most of the high-level logic.

1. **Kernel Source Code**

Let’s just check out the source code and look for anything that might help. There is a “reset to factory settings” button on the router. This button is part of the hardware layer, which means the GPIO pin that detects the button press must be controlled by the drivers. These are the logs we saw coming out of the UART port in a previous post:

1. **Binary Disassembly**

The code inside every executable binary is just a compilation of instructions encoded as Machine Code so they can be processed by the CPU. Due to the very low-level nature of the kernel, and how heavily it interacts with the hardware, it is incredibly difficult to make any sense of its binary. Userspace binaries, on the other hand, are abstracted away from the hardware and follow unix standards for calling conventions, binary format, etc. They’re an ideal target for disassembly.

1. **A little bit about disassemblers**

In order to display the assembly code in a more readable way, all these disasemblers use a “Graph View”. It provides an intuitive way to follow the different possible execution flows in the binary:

1. **Example of User Space Binary Disassembly**

Following up on the reset key example we were using for the Kernel, we’ve got the code that generated **some** of the UART log messages, but not all of them. Since we couldn’t find the ‘button has been pressed’ string in the kernel’s source code, we can deduce it must have come from user space. Let’s find out which binary printed it:

~/Tech/Reversing/Huawei-HG533\_TalkTalk/router\_filesystem

$ grep -i -r "restore default success" .

Binary file ./bin/cli matches

Binary file ./bin/equipcmd matches

Binary file ./lib/libcfmapi.so matches

3 files contain the next string found in the logs: 2 executables in /bin/ and 1 shared object in /lib/. Let’s take a look at /bin/equipcmd with IDA:

If we look closely, we can almost read the C code that was compiled into these instructions. We can see a “clear configuration file”, which would match the ERASE commands we saw in the SPI traffic capture to the flash IC. Then, depending on the result, one of two strings is printed: restore default success or restore default fail. On success, it then prints something else, flushes some buffers and reboots; this also matches the behaviour we observed when we pressed the reset button.

That’s about all

1. **Questions**

* Have you ever disassembled anything?

Yes, in the 10th grade I disassembled the game Counter Strike and cheats for it. In the spring of this year I experimented with viruses on a virtual machine with Kali linux and Ubuntu linux

* What programs have you used for reverse engineering?

I used 3 programs. These are IDA Pro, Radare2, Binary Ninja

1. **Thanks for watching**