Kick-Off P2

Daniel Kales & Peter Peßl

Information Security – WT 2019/20

SCIENCE
PASSION
TECHNOLOGY

Organizational

- We have some solo groups after the first assignment
- If you want to be merged with another solo group...
  - ... come down to us after this lecture
  - ... send me a mail today!
- We will try to merge groups with similar point total

Kick-off for P2: System-Security

Bugs in Software and Hardware

P2: Overview

- 2 main categories:
  - Hacklets
  - Faults
- Your task:
  - Hacklets: exploit common errors in C ...
  - Faults: use (simulated) physical attacks ...
- ... to recover secret information

P2: Timeline

- Kickoff - Now
- “My first exploit” tutorial - 15.11.2019, 13:30
- Fault demo & Question hour - 22.11.2019, 13:30
- Question hour - 29.11.2019, 13:30
- **Deadline** - 06.12.2019, 23:59

P2: Assignment

- Detailed specification on a seperate assignment sheet
  - Available on course website
  - Read both the assignment sheet and these slides!
- Submission and file-distribution using git
  - use the same-repository (P2 subfolder)
  - pull the assignment files from the upstream repository
    - see course website for instructions!
- Points will be published online
  - Automated test system with daily tests for each task
  - Links on course website

P2: Assignment

- Detailed specification on a seperate assignment sheet
  - Available on course website
  - Read both the assignment sheet and these slides!
- Submission and file-distribution using git
  - use the same-repository (P2 subfolder)
  - pull the assignment files from the upstream repository
    - see course website for instructions!
- Points will be published online
  - Automated test system with daily tests for each task
  - Links on course website

P2: Assignment

- Detailed specification on a seperate assignment sheet
  - Available on course website
  - Read both the assignment sheet and these slides!
- Submission and file-distribution using git
  - use the same-repository (P2 subfolder)
  - pull the assignment files from the upstream repository
    - see course website for instructions!
- Points will be published online
  - Automated test system with daily tests for each task
  - Links on course website

P2: Framework

You will get a VM

- All tools are pre-installed
- Do not use additional libraries, etc...

Where should you begin?

- Download the VM
- Setup the VM
- Clone the assignment from the upstream repo
- Read the task description, read the hints

P2: Framework

You will get a VM

- All tools are pre-installed
- Do not use additional libraries, etc...

Where should you begin?

- Download the VM
- Setup the VM
- Clone the assignment from the upstream repo
- Read the task description, read the hints

Hacklets

Exploiting Common Software Errors

Overview

- For the hacklet task:
  - Analyze 7 small C and C++ programs
  - Find mistakes in the programs
  - Exploit these mistakes
  - Capture the flag (contents of a `flag.txt` file)
- Convince the program to give you the flag
  - Write an exploit using python3 (no actual C programming needed!)
    - But you need to understand the C source to find mistakes!
  - Print the flag to stdout and store it to `solution.txt`

Overview

- For the hacklet task:
  - Analyze 7 small C and C++ programs
  - Find mistakes in the programs
  - Exploit these mistakes
  - Capture the flag (contents of a `flag.txt` file)
- Convince the program to give you the flag
  - Write an exploit using python3 (no actual C programming needed!)
    - But you need to understand the C source to find mistakes!
  - Print the flag to stdout and store it to `solution.txt`

Where do I begin?

- Take a look at the hacklets
- Analyze the source code
- Use GDB to debug the hacklets
- Execute the hacklets, test different inputs
- Test strange input
- Does the code behave like it should?

What kind of vulnerabilities will we find?

For example, and in no particular order:

- Format String Vulnerabilities

```
char user_input[10];
...
<read user input>
...
printf(user_input);
```

What kind of vulnerabilities will we find?

For example, and in no particular order:

- Buffer Overflows

```
char numbers[10];
...
printf("%d", numbers[10]);
...
numbers[100] = 17;
```

What kind of vulnerabilities will we find?

For example, and in no particular order:

- Use After Free

```
char* temp = malloc(10);
...
free(temp);
...
printf("%s", temp);
```

What is a valid solution?

- A file called `exploit` (already present in each folder) containing a python 3 script that exploits the main.elf such that
  - you get the flag (contents of `flag.txt`)
  - the flag is printed to stdout and/or stored to `solution.txt`

- Stuff to keep in mind
  - We will test with a different, random flag
  - The size of the flags can vary
  - We will test with the original main.elf
  - You should never hardcode the flag!

What is a valid solution?

- A file called `exploit` (already present in each folder) containing a python 3 script that exploits the main.elf such that
  - you get the flag (contents of `flag.txt`)
  - the flag is printed to stdout and/or stored to `solution.txt`

- Stuff to keep in mind
  - We will test with a different, random flag
  - The size of the flags can vary
  - We will test with the original main.elf
  - You should never hardcode the flag!

Contact & Finding Help

- Course website: `https://www.iaik.tugraz.at/infosec`
- `infosec@iaik.tugraz.at`
- If you need help for the exercises, try (in this order):
  - Newsgroup `graz.lv.infosec`
    - Don’t post your solution there...
  - Contact the responsible teaching assistant
  - Contact the responsible lecturer for the practicals
- Come to the question hours

Faults

It’s only secure if executed correctly

We want to build a secure program…

- We use proven cryptography
  - use standardized and highly scrutinized algorithms
  - use implementation from a secure library
  - avoid misuse (proper randomness, AEAD, …)
  - …

- We avoid or detect programming mistakes
  - address sanitization, stack canary, ASLR, …
  - use „memory-safe“ programming language
  - …

Are we secure?

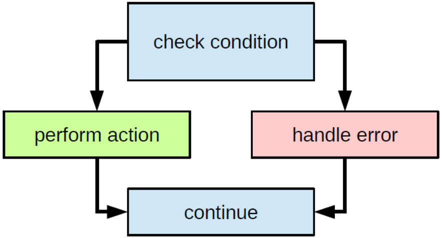
- Some additional requirements, such as:

The program is executed correctly /
The processor works as intended

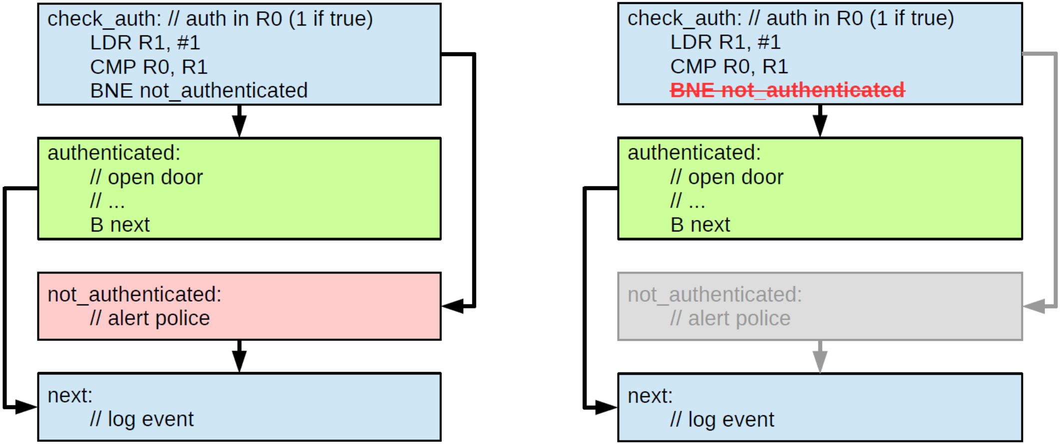
- What happens when it doesn‘t? What if it…
  - „forgets“ to execute certain instructions
  - performs incorrect computations, such as 2*3 = 4
  - forgets data (memory reliability)

Example: PIN check

```
unsigned pin = read_pin();
bool auth = tpm_check(pin);
if( auth ) {
	open_door();
} else {
	alert_police();
}
log_event();
```



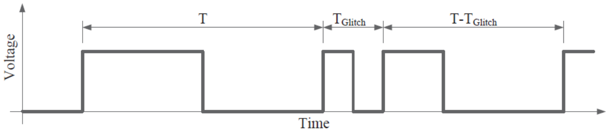
Example: PIN check

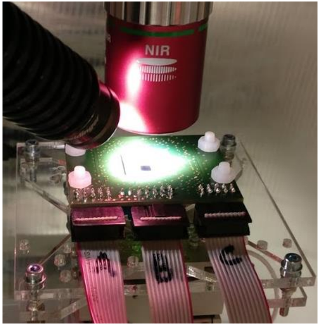


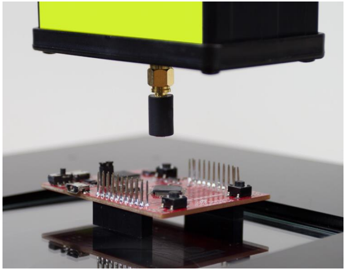
The Setting of Fault Attacks

- CPUs work correctly as long as operated within specification
  - datasheet: supply voltage, clock speed, ambient temperature, etc.
- Problem: attacker can have physical access to device
  - ex: stolen banking card
- Attacker does not care about specification
  - carefully manipulate device to force errors (faults)

Means of Faulting







- Supply voltage spikes
- Clock glitching
- EM transient injections
- Laser
- …

Results of Faulting

- Possible faults
  - skip instructions, incorrect computations, memory corruption
- Exploitation
  - bypass security checks, disable countermeasures, recover cryptographic keys…
  - **We want you to try that!**
- Problem: we don‘t have enough lasers for everyone

Fault Simulator

- For exploitation: don‘t care how fault is injected
  - important: just its effect

- We give you a **Fault Simulator**
  - lets you inject typical faults into execution of any binary
  - configuration: specify which kind of fault you want to inject (and when)

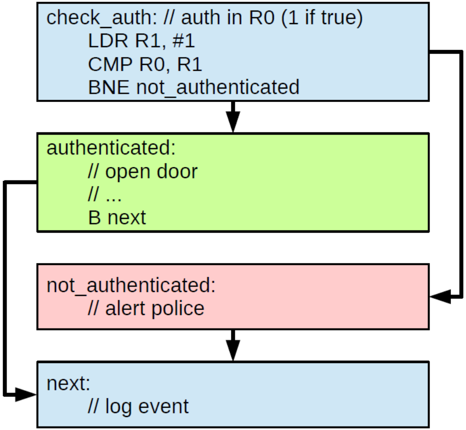
- Examples:
  - „skip the 1495th ASM instruction after startup“
  - „flip bit at adress 0xbeef when instruction pointer is 0xdead“

Your Task

- 3 challenges: attack precompiled binaries with our simulator
- One or two steps
  1. **Specify your faults**
     - for each challenge, we restrict allowed number of faults and their type
  2. **Perform post-processing of faulty outputs (Python3 script)**
     - sometimes faulting alone is not enough, need post-processing of outputs
     - ex: fault encryption, such that comparing faulted and correct output lets you recover key

Challenge: 01_password

- Bypass a password check
- using a single **instruction skip**



Challenge: 02_eddsa

- Problem: nonce reuse
  - same nonce for different messages → key recovery (see P1)

- Solution: make nonce generation deterministic
  - n = Hash(m|h), where h is secret
  - same nonce for different messages would mean hash collision

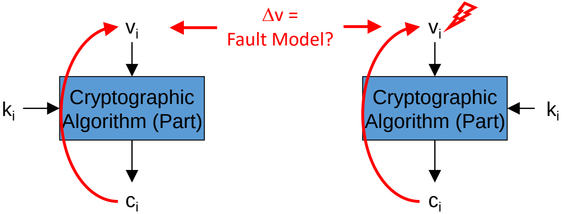
- Problem: achieving „nonce reuse“ is easy now
  - But can you sign a different message with the reused nonce?

Challenge: 03_aes

- Fault attacks on symmetric crypto: more tricky

- Differential Fault Attack
  - compare faulty and real output
  - compute back to key

- You can **flip bits** (very precisely)



Framework

- Similar to P1 and hacklets
  - Each challenge in separate folder
  - Python scripts with provided helper functions and section for your code
- Secrets
  - locally: you can access secrets, for developing, testing, debugging, etc.
  - test system: new set of secrets, access is locked
- Important: solution for **unmodified** binary
  - modifications for testing of course possible

More Information

- Assignment sheet
- Readme of fault simulator
- Demo exploits
  - examples for fault simulator

- Lecture next week
- Tutorial with live demo of fault attack on microcontroller
- Question hours

Questions