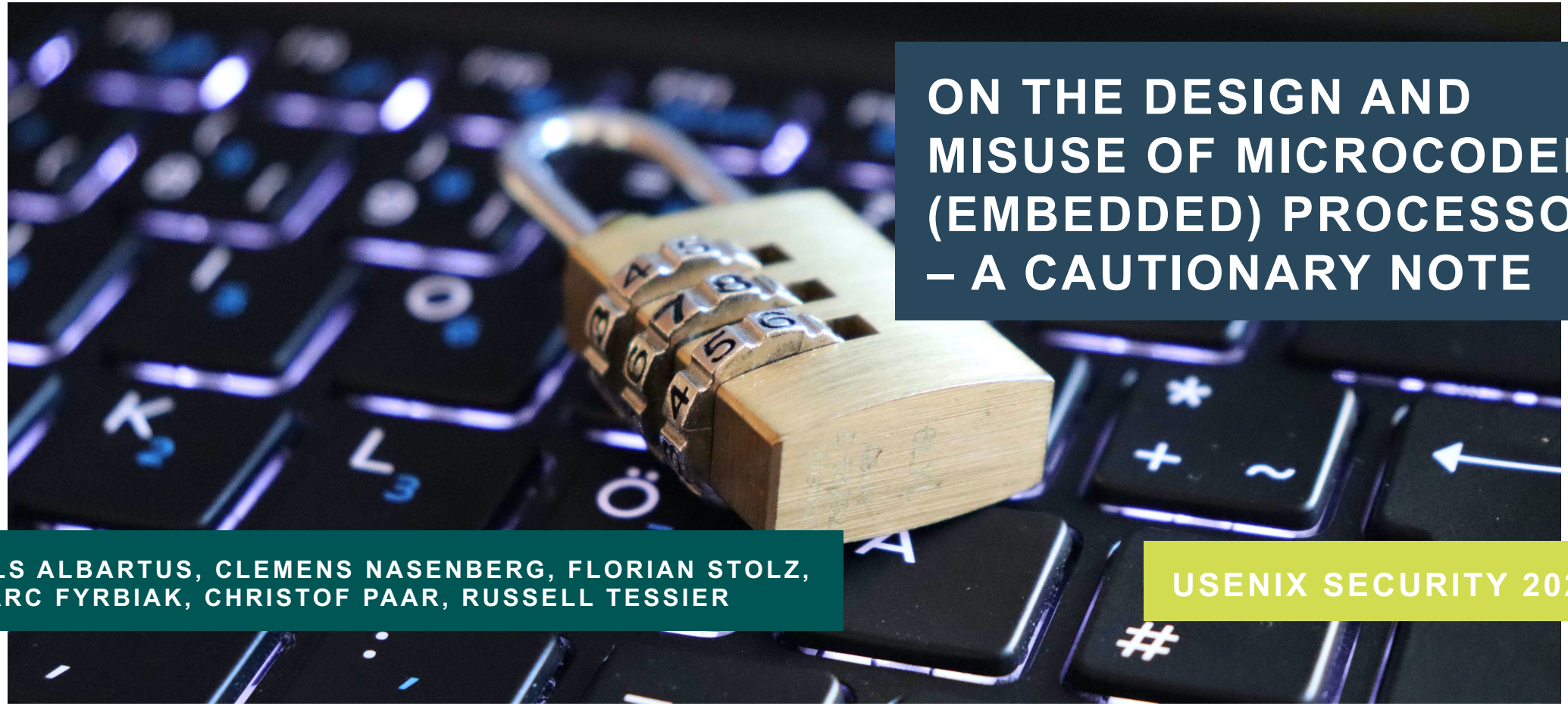
UMass Amherst

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

RUHR
UNIVERSITÄT
BOCHUM
RUB



ON THE DESIGN AND MISUSE OF MICROCODED (EMBEDDED) PROCESSORS – A CAUTIONARY NOTE

NILS ALBARTUS, CLEMENS NASENBERG, FLORIAN STOLZ,
MARC FYRBIAK, CHRISTOF PAAR, RUSSELL TESSIER

USENIX SECURITY 2021

MOTIVATION – FDIV BUG INTEL

- Bugs in hardware/silicon can generally not be fixed
  - Famous example: FDIV bug

- Today: Small updatable part in the heart of the CPU (Microcode)

WHAT DO WE KNOW ABOUT COMMERCIAL MICROCODE?

Sadly…not too much

- **Only few information are available from the vendors**

Updates are issued on a regular basis by Intel and AMD

- **Cryptographically protected**
- **→ In-depth microcode analysis is not possible**

Summary of Intel Microcode Updates

Windows Server 2019, all editions, Windows 10, version 1809, all editions, More...

Intel microcode updates

Microsoft is making available Intel-validated microcode updates that are related to Spectre Variant 2 (CVE 2017-5715 ["Branch Target Injection"]).

The following table lists specific Microsoft Knowledge Base articles by Windows version. The article contains links to the available Intel microcode updates by CPU:

KB Number and Description	Windows Version	Source
KB4100347 Intel Microcode Updates	Windows 10, version 1803, and Windows Server, version 1803	Windows Update, Windows Server Update Service, and Microsoft Update Catalog
KB4090007 Intel Microcode Updates	Windows 10, version 1709, and Windows Server 2016, version 1709	Windows Update, Windows Server Update Service, and Microsoft Update Catalog
KB4091663 Intel Microcode Updates	Windows 10, version 1703	Windows Update, Windows Server Update Service, and Microsoft Update Catalog
KB4091664 Intel Microcode Updates	Windows 10, version 1607, and Windows Server 2016	Windows Update, Windows Server Update Service, and Microsoft Update Catalog
KB4091666 Intel Microcode Updates	Windows 10 (RTM)	Windows Update, Windows Server Update Service, and Microsoft Update Catalog

Even Spectre was fixed via microcode updates

Research Question:

What is the threat potential of malicious microcode updates?

MICROCODE UPDATES

Recently: Extraction of secret decryption key for Intel CPUs [1, 2]

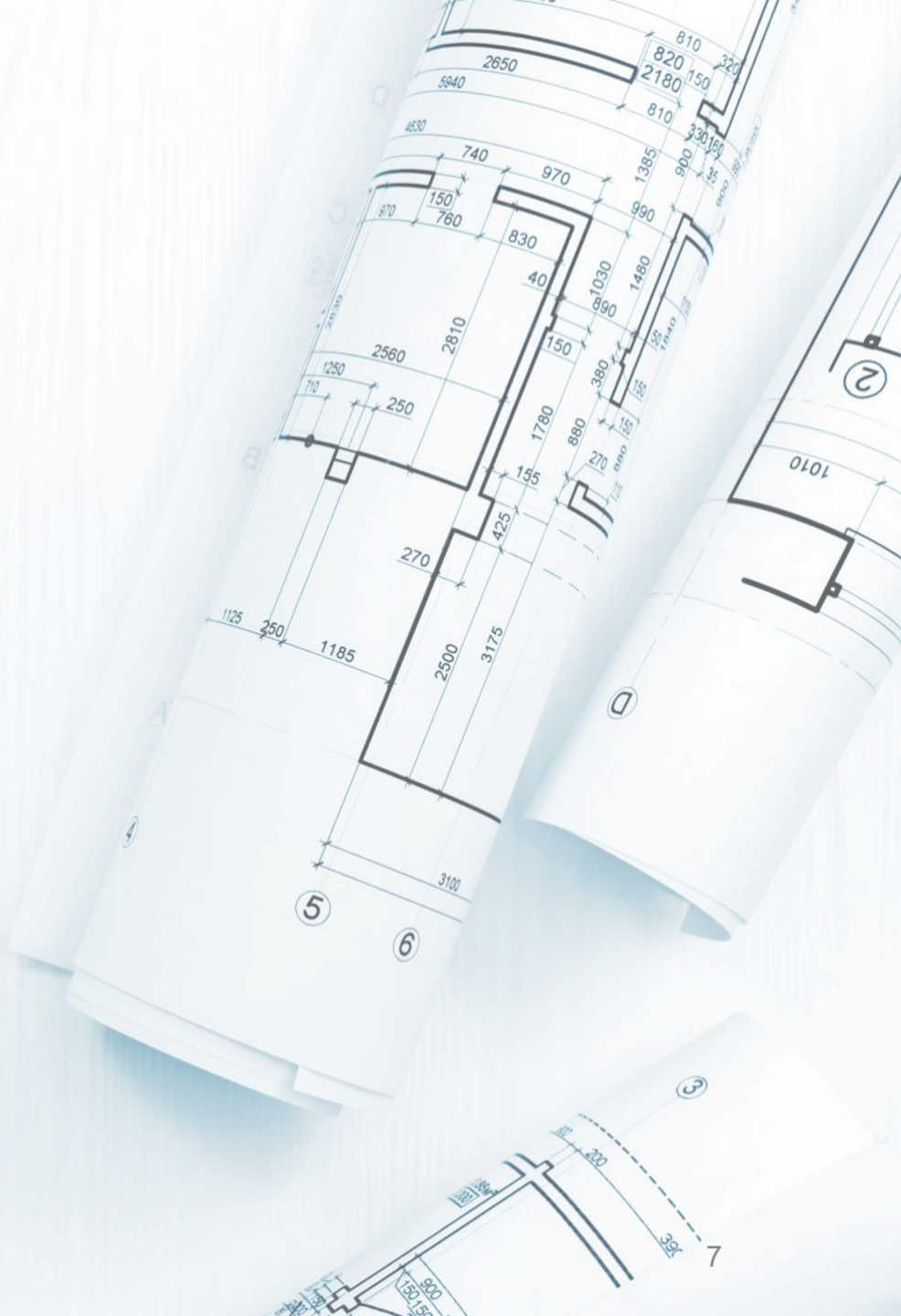
But…

- **Microcode format still needs to be reverse engineered**
  - **Partly done by researchers [3, 4]**

- **Physical access is necessary, since the signing keys are not affected**

[1] https://github.com/chip-red-pill/glm-ucode
[2] https://arstechnica.com/gadgets/2020/10/in-a-first-researchers-extract-secret-key-used-to-encrypt-intel-cpu-code/
[3] Reverse Engineering x86 Processor Microcode, Koppe et al 2017, USENIX Security
[4] https://github.com/chip-red-pill/uCodeDisasm

BUILDING A MICROCODE EVALUATION FRAMEWORK



MICROCODE EVALUATION PLATFORM FOR RISC-V

- Simple microcoded (embedded) RISC-V I
  - Von Neumann architecture

- No instruction parallelism (no pipeline etc.)
- Bare-metal software execution (no OS)

- Microcode generator to easily prototype new instructions (and our Trojans)

HOW DOES MICROCODE – ON OUR CPU – WORK?

- **One microprocessor instruction (macroinstruction) is broken down in a set of microinstructions**

```
ADD rd, rs1, rs2
```

```
1  def add
2      add0: a <- rf[rs1];
3      add1: b <- rf[rs2];
4      add2: rf[rd] <- alu(a + b); fetch;
```

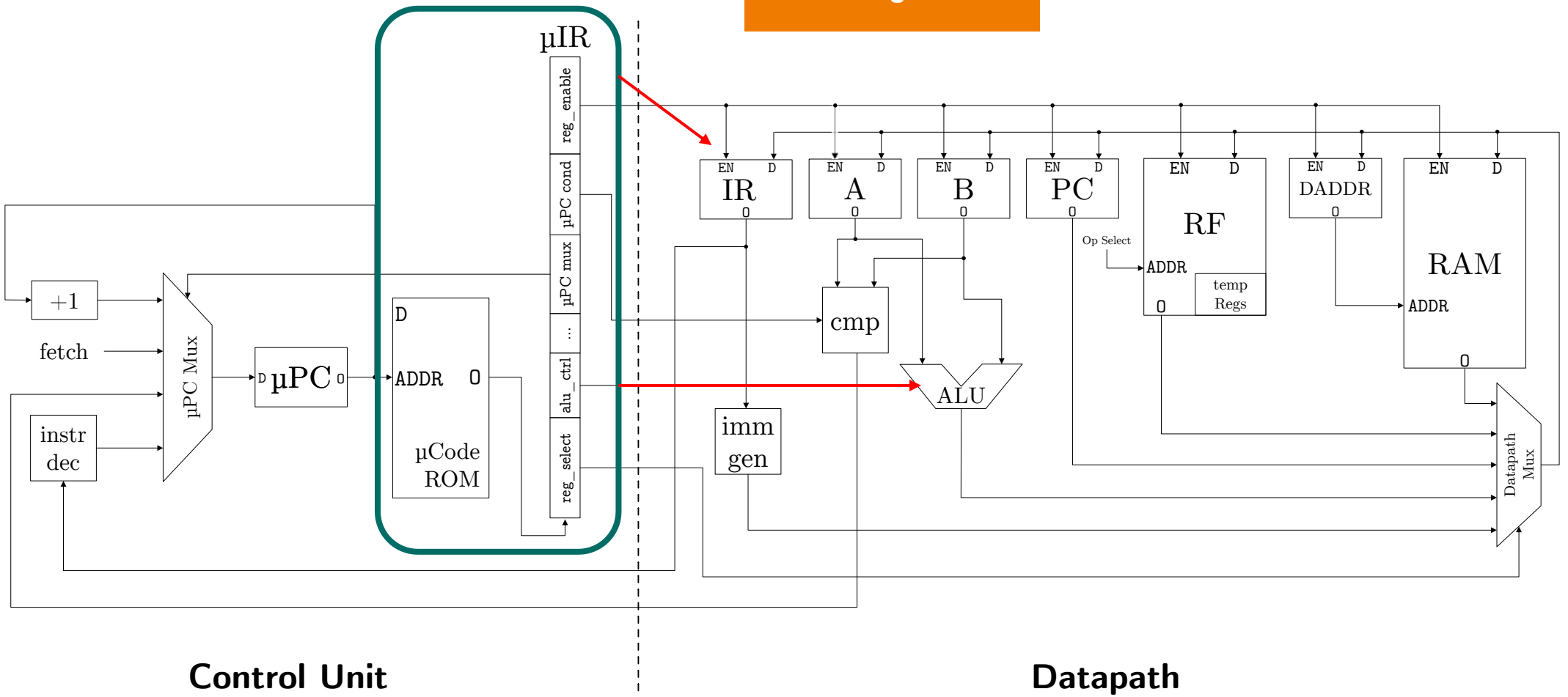
Microcode Generator

```
1  0000000101  //a0
2  0000200201  //a1
3  0008406020  //a2
```

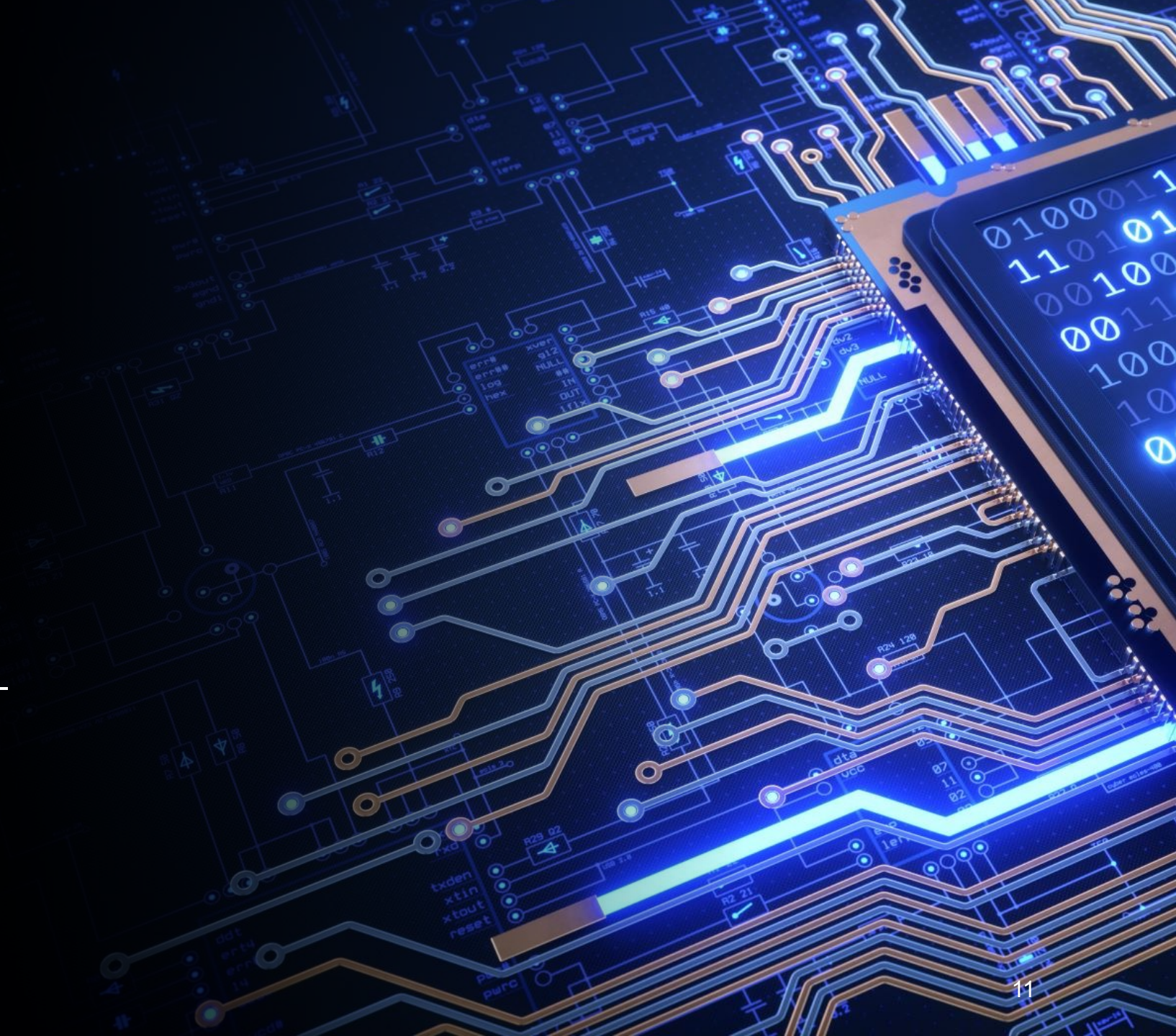
Generated control signals, which can be inserted into the microcode ROM

MICROCODED RISC-V

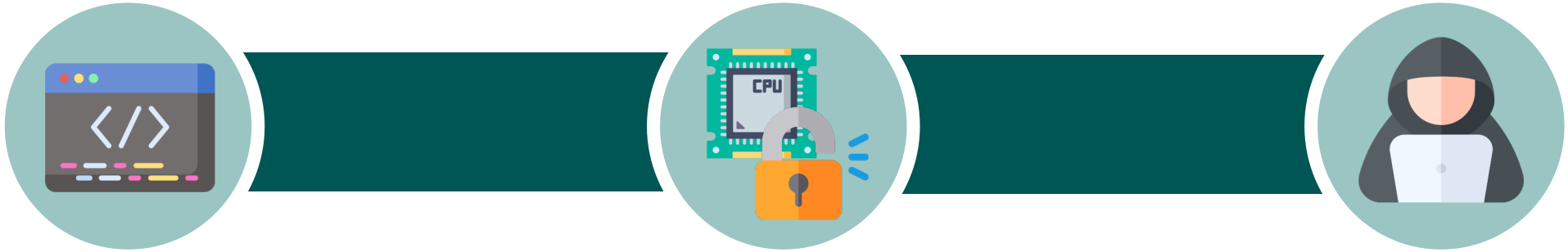
Microcode stores control signals



BUILDING MICROCODE TROJANS



ATTACKER CAPABILITIES



Manipulation of the behavior of instructions

- … or even add new custom instructions

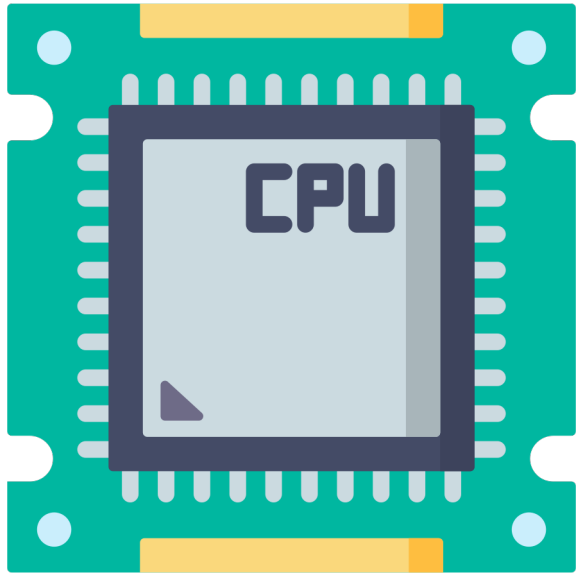
Access to general-purpose registers

- PC, IR, GPR…
- Microcode scratch registers

Turing-complete capabilities

- Control of data-path (incl. ALU)
- Conditional branches

(SEEMINGLY) UNLIMITED CAPABILITIES VS LIMITED INFORMATION



XOR X13, X11, X2

Problems:

- **What is happening on a higher-level?**
- **Limited due to size of Microcode ROM**

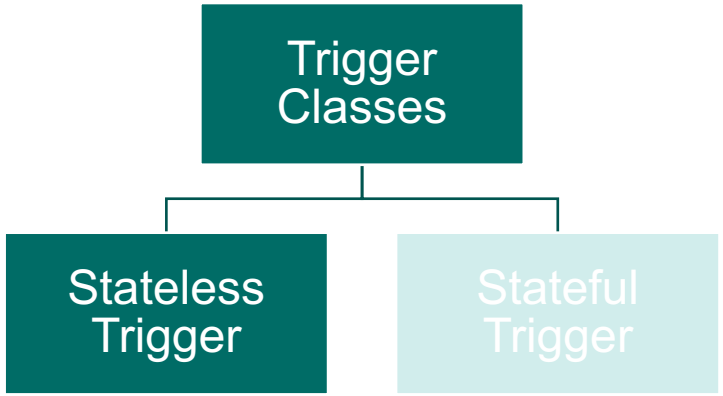
TRIGGER CLASSES
STATELESS TROJANS

- **Do not possess any kind of state/memory and can only see the currently executed instructions**

- **Simple Trigger Conditions**

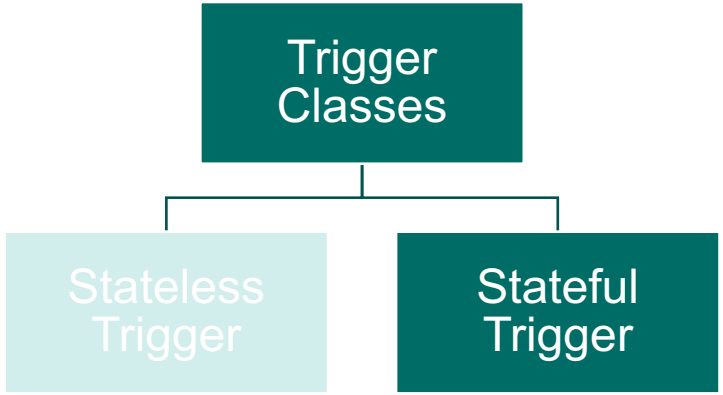
```
ADDI X0, X1, 0xbadbad
```

- **Can combine multiple CPU registers and statuses**
  - e.g.: IR, PC, General Purpose Registers…



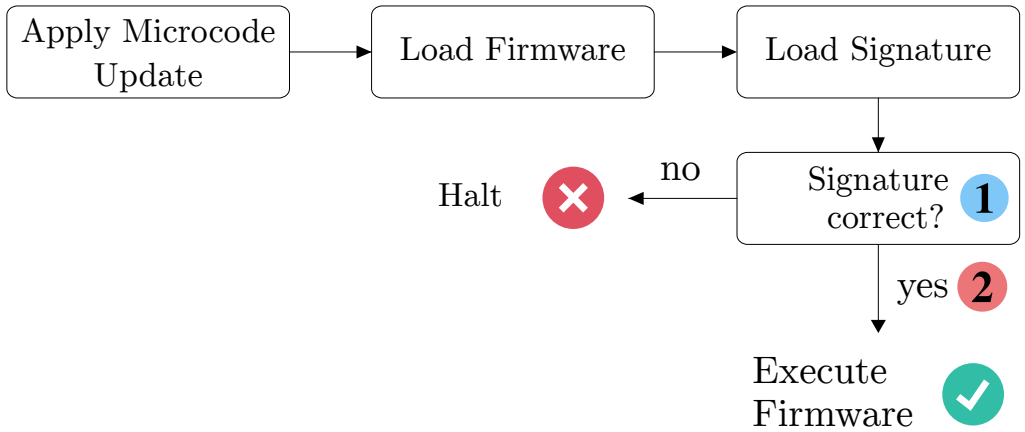
TRIGGER CLASSES
STATEFUL TRIGGERS

- **Possess state that spans across multiple instructions**
- **Save information in, e.g. scratch registers; Allows for implementation of counters, state machines and more**
- **Can be used to match for certain instruction sequences**
- **Allow for (limited) high-level information, confined by the size of the microcode ROM**



TROJAN CASE STUDY

SIMPLIFIED BOOT PROCESS OF SECURE BOOT



Disclaimer: Extremely simplified high-level view. No industry standard exists, so implementation is vendor dependent

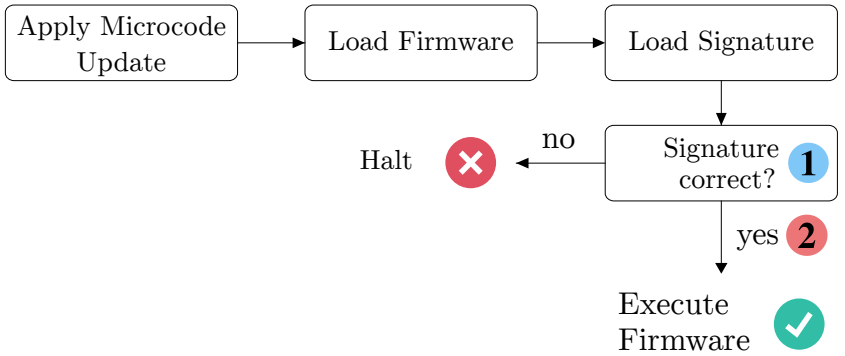
SIMPLIFIED BOOT PROCESS OF SECURE BOOT

Setup and Assumptions:

- **Attacker has access to the boot-code binary in ROM**
- **Secure boot code cannot change, but microcode can**

High-Level Attack Idea:

- **Bypass cryptographic checks**
- **Manipulate program flow to always take the jump that executes the firmware**



HIGH-LEVEL ATTACK IDEA

```
1  if (verify(firmware, signature)){
2    asm volatile("jal ra, 0x7000");
3  } else {
4    while (1) {} // trap CPU
5  }
```

```
1  0x230: jal   ra,280 <verify>
2  0x234: addi  a5,a0,0
3  0x238: beq   a5,zero,240 <main+0x240>
4  0x23c: jal   ra,7000 <_isatty+0x20>
5  0x240: jal   zero,240 <main+0x240>
```

1 Trigger:

- **BEQ instruction at address `0x238`**
  - **Performs the check if signature is valid**

BEQ *a5*, zero, *240*

2 Trojan Payload:

- **Always take the jump that executes the firmware**

ATTACK DETAILS

ATTACK (1)

- **Signature matches firmware**
  - **verify() succeeds**

```
0x230: jal  ra,280 <verify>
0x234: addi a5,a0,0
0x238: beq  a5,zero,240 <main+0x240>
0x23c: jal  ra,7000 <_isatty+0x20>
0x240: jal  zero,240 <main+0x240>
```

verify check

execute firmware

BEQ *rs1, rs2, dst*

```
def beq
    beq0: a <- rf[rs1];
    beq1: b <- rf[rs2];
    beq2: b <- ig(imm_b); if a != b fetch;
    beq3: b <- ig(imm_b);
    beq4: a <- pc;
    beq5: a <- alu(a - 4);
    beq6: pc <- alu(a + b);
    beq7: fetch;
```

Compares `rs1, rs2` and `fetches` next instruction if they don't match (true case)

ATTACK (2)

- **Signature does not match firmware**
  - **verify() fails**

```
1  0x230: jal   ra,280 <verify>
2  0x234: addi  a5,a0,0
3  0x238: beq   a5,zero,240 <main+0x240>   <-- verify check
4  0x23c: jal   ra,7000 <_isatty+0x20>
5  0x240: jal   zero,240 <main+0x240>      <-- trap CPU
```

BEQ *rs1, rs2, dst*

```
1  def beq
2      beq0: a <- rf[rs1];
3      beq1: b <- rf[rs2];
4      beq2: b <- ig(imm_b); if a != b fetch;
5
6                                   Adds offset to PC and
7                                   jump to false case
8
9
10
11     beq3: b <- ig(imm_b);
12     beq4: a <- pc;
13     beq5: a <- alu(a - 4);
14     beq6: pc <- alu(a + b);
15     beq7: fetch;
```

ATTACK (3)

- **Signature does not match firmware**
  - **verify() fails**
  - **…but microcode Trojan will execute the firmware**

```
0x230: jal  ra,280 <verify>
0x234: addi a5,a0,0
0x238: beq  a5,zero,240 <main+0x240>
0x23c: jal  ra,7000 <_isatty+0x20>
0x240: jal  zero,240 <main+0x240>
```

verify check

execute firmware

BEQ *rs1, rs2, dst*

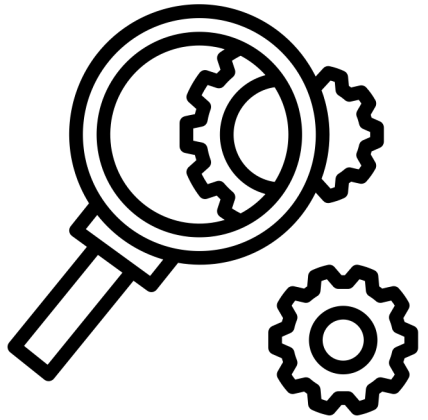
```
def beq
    beq0: a <- rf[rs1];
    beq1: b <- rf[rs2];
    beq2: b <- ig(imm_b); if a != b fetch;
    beq3: b <- ig(imm_b);
    beq4: a <- pc;
    beq5: a <- alu(a - 4);
    beq6: pc <- alu(a + b);
    beq7: fetch;
```

Trojan checks current address of PC and fetches next instruction

CONCLUSION

CONCLUSION

- Microcode Trojans provide a powerful attack vector
- Microcode Trojans have to be custom-tailored to the regarding CPU architecture and application
- In the paper:
  - More Trojans
  - Discussion on microcode Trojans on more advanced systems
  - Possible Mitigations



THANK YOU FOR YOUR ATTENTION

You can contact us via mail:

nils.albartus@mpi-sp.org

We are always looking for interesting new projects ☺