CustomProcessingUnit: Reverse Engineering and Customization of Intel Microcode

WOOT 2023

Pietro Borrello
Sapienza University of Rome

Catherine Easdon
Dynatrace Research & Graz University of Technology

Martin Schwarzl
Graz University of Technology

Roland Czerny
Graz University of Technology

Michael Schwarz
CISPA Helmholtz Center for Information Security
The first CPU µcode Software Framework

- µcode Static analysis
- µcode Dynamic analysis
How do CPUs work?
• Red Unlock of Atom Goldmont (GLM) CPUs
• Extraction and reverse engineering of GLM μcode format
• Discovery of undocumented control instructions to access internal buffers
Two secret instructions that can access:

- System agent
- URAM
- Staging buffer
- I/O ports
- Power supply unit
- CRBUS
CPU interacts with its internal components through the CRBUS
- MSR → CRBUS addr
- Control and Status registers
- Post Silicon Validation features
What can you do with access to microarchitectural buffers?
Microcoded Instructions 101

... cpuid ...

XLAT

\[ \mu \text{code ROM} \]
Microcoded Instructions 101

XLAT

\[
\begin{array}{c}
\ldots \\
cpuid \\
\ldots \\
\ldots \\
\ldots \\
\ldots \\
\end{array}
\]

\[
\begin{array}{c}
\mu\text{code ROM} \\
\mu\text{code RAM} \\
\end{array}
\]
Microcoded Instructions 101

... cpuid ...

XLAT

... match & patch ...

μcode ROM

μcode RAM

Pietro Borrello (@borrello_pietro)
Building a Ghidra µcode Decompiler

Pietro Borrello (@borrello_pietro)
Building a Ghidra µcode Decompiler

void rc4_decrypt(ulong tmp0_i, ulong tmp1_j, byte *ucode_patch_tmp5, int len_tmp6, byte *S_tmp7, long callback_tmp8)
{
    byte bVar1;
    byte bVar2;

    do {
        tmp0_i = (ulong)(byte)((char)tmp0_i + 1);
        bVar1 = S_tmp7[tmp0_i];
        tmp1_j = (ulong)(byte)(bVar1 + (char)tmp1_j);
        /* swap S[i] and S[j] */
        bVar2 = S_tmp7[tmp1_j];
        S_tmp7[tmp0_i] = bVar2;
        S_tmp7[tmp1_j] = bVar1;
        *ucode_patch_tmp5 = S_tmp7[(byte)(bVar2 + bVar1)] ^ *ucode_patch_tmp5;
        ucode_patch_tmp5 = ucode_patch_tmp5 + 1;
        len_tmp6 += -1;
    } while (len_tmp6 != 0);
    /*(code *)(callback_tmp8 * 0x10))();
    return;
}
Accessing the µcode

Reverse engineer how the CPU itself updates µcode
  • Observe patterns of CRBUS accesses
  • Reproduce the same accesses using the undocumented instructions
→ With the undocumented instructions we can control µcode!
Leveraging udbgrd/wr we can patch μcode via software

- Completely **observe** CPU behavior
- Completely **control** CPU behavior
- All within a BIOS or **kernel** module
μcode Framework

Patch μcode

Hook μcode

Trace μcode

Pietro Borrello (@borrello_pietro)
We can customize the CPU’s behavior.

- **Change** microcoded instructions
- **Add** functionalities to the CPU
Improve CPU security and performance through μcode customization

- x86 Pointer Authentication Codes
- Fast Breakpoints
- Constant Time Hardware Division
Install µcode hooks to observe events.

- Setup Match & Patch to execute custom µcode at certain events
- Resume execution
μcode traces

Trace μcode execution leveraging μcode hooks.

- Setup a hook for every possible μop
- Reconstruct μops executed
GLM μcode update algorithm

- `wrmsr` → move ucode patch to 0xfeb01000
- `SHA256` → check
- `CPU secret` → key expansion
- `RC4 key` → decrypt
- `SHA256` → RSA verify
- `nonce` → metadata, RSA mod, RSA exp, RSA sig, ucode patch
- `discard first 0x200 bytes`
A μcode update is bytecode: the CPU interprets commands from the μcode update.

- reset
- write μcode
- hook match & patch
- write stgbuf
- write uram
- CRBUS cmd
- control flow directives
- nested decrypt (e.g., XuCode)
- Create a parser for µcode updates
- Automatically collect existing µcode(s) for GLM
- Decrypt all GLM updates

github.com/pietroborrello/CustomProcessingUnit/ucode_collection
Conclusion

• Deepen understanding of modern CPUs with $\mu$code access
• Develop a static and dynamic analysis framework for $\mu$code:
  • $\mu$code decompiler
  • $\mu$code assembler
  • $\mu$code patcher
  • $\mu$code tracer
• Let’s control our CPUs!

github.com/pietroborrello/CustomProcessingUnit