All you ever wanted to know about the AMD Platform Security Processor and were afraid to emulate

INSIDE A DEEPLY EMBEDDED SECURITY PROCESSOR.
Outline

• What is the Platform Security Processor (PSP)?
  • Why emulate it?
• How to emulate the PSP
• What can we do with the emulator?
**AMD SECURE PROCESSOR**

A Dedicated Security Subsystem

- **AMD Secure Processor integrated within SoC**
  - 32-bit microcontroller (ARM Cortex-A5)
- Runs a secure OS/kernel
- Secure off-chip NV storage for firmware and data (i.e. SPI ROM)
- Provides cryptographic functionality for secure key generation and key management
- Enables hardware validated boot

**Hardware Root of Trust Provides Foundation for Platform Security**

---

1 Formerly known as Platform Security Processor (i.e. PSP)

Server & Desktops (Epyc & Ryzen)

Integrated since 2013

Undocumented, proprietary firmware

Acts as trust anchor
Why Emulate?

• Proprietary software at the highest privilege level
• Static analysis is possible but time consuming (boring 😊)
  • Only good for a single firmware version
• Emulation (if done right) enable easy analysis of future firmware versions
PSPTOOL

Python-based

Command-line interface

Parsing

Extraction

Decompression

Manipulation

PEM export of keys

Signature verification

Duplicate detection

Signature update

Python API

GPLv3

PSPTool is a Swiss Army knife for dealing with firmware of the AMD Secure Processor (formerly known as Platform Security Processor or PSP). It locates AMD firmware inside UEFI images as part of BIOS updates targeting AMD platforms.

It is based on reverse-engineering efforts of AMD’s proprietary filesystem used to pack firmware blobs into UEFI Firmware Images. These are usually 16MB in size and can be conveniently parsed by UEFITool. However, all binary blobs by AMD are located in padding volumes unparsable by UEFITool.

PSPTool favourably works with UEFI images as obtained through BIOS updates.

Installation

You can install PSPTool either through pip,

```
pip install psptool
```
Uncover, Understand, Own - Regaining Control Over Your AMD CPU

https://media.ccc.de/v/36c3-10942-uncover_understand_own_-_regaining_control_over_your_amd_cpu
**BOOT PROCESS: EPYC**

- PSP boots *before* the x86 cores
- **On-Chip** Bootloader loads **Off-Chip** bootloader from flash
- **Off-Chip** Bootloader loads and executes apps in specific order
- System is initialized by different **ABL stages**
- Load UEFI image and release x86 cores from reset
- **SEV app** is loaded during runtime upon the request of the OS
BOOT PROCESS: RYZEN

• PSP boots *before* the x86 cores
• *On*-Chip Bootloader loads *Off*-Chip bootloader from flash
• *Off*-Chip Bootloader loads and executes apps in specific order
• System is initialized by different ABL stages
• Load UEFI image and release *x86* cores from reset
**BOOT PROCESS: RYZEN**

- PSP boots *before* the x86 cores
- **On-Chip** Bootloader loads **Off-Chip** bootloader from flash
- **Off-Chip** Bootloader loads and executes apps in specific order
- System is initialized by different **ABL stages**
- Load UEFI image and release *x86* cores from reset
- **SRAM is overwritten with Secure OS (Kinibi TEE)**
**BOOT PROCESS: RYZEN**

- PSP boots *before* the x86 cores
- **On-Chip** Bootloader loads **Off-Chip** bootloader from flash
- **Off-Chip** Bootloader loads and executes apps in specific order
- System is initialized by different **ABL stages**
- Load UEFI image and release **x86** cores from reset
- **SRAM is overwritten with Secure OS (Kinibi TEE)**
  - Firmware TPM is *one* application of this OS
PSP MEMORY LAYOUT

• 256 KB (Zen1) or 384 KB (Zen2) SRAM
  • Off-Chip BL and Applications
• On-Chip BL (ROM) at ARM high vectors (0xFFFF0000)
• MMIO: IRQ controller (custom), timer, crypto accelerator (CCP), X86 and SMN slot controller
• System Management Network Slots
• X86 address space slots

A slot is a “view” into another address space
PSP ADDRESS SPACES

PSP

- SRAM
- SMN Slots
- PSP MMIO
- X86 Slots
- ROM Bootloader

System Management Network

- SPI Flash
- Fuses
- Memory controller (with encryption slots)

32 Bit

32 Bit
PSP ADDRESS SPACES

PSP

- SRAM
- SMN Slots
- PSP MMIO
- X86 Slots
- ROM Bootloader

x86 address space

- MMIO
- DRAM
- PCI
- Memory mapped IO ports (0x3f8 ...)

32 Bit

48 Bit

Documented (partially): Processor Programming Reference (PPR)
PSP CRYPTO ACCELERATOR (CCP)

- PSP contains a Cryptographic Coprocessor V5 (CCP)
- Support for: SHA, RSA, AES, ECC, ZLIB, TRNG
- Used to verify signatures, decompress firmware files and as a DMA copy engine
- No “official” documentation available, but....

There is a Linux kernel driver: drivers/crypto/ccp
WHILE (!SUCCEEDED) {

Analyze -> Implement -> Execute -> Fail

Implement

Execute

Analyze

Fail

+337 -2

}
SUCCESS! (KIND OF)

- On Chip BL completes
- Off Chip BL starts and executes first two apps
- Off Chip BL executes first ABL stage but what next?
- Emulating all devices not feasible
INFO STS 0x000057c6[0x0001165] SVC, S, M, I,NF,0x00014000 STRING "POST CODE (PSP): PSPSTATUS_C2P_MASTER_INITIALIZED_SLAVE_WAITED_FOR_MASTER"
INFO STS 0x000057c6[0x000122f] SVC, S, M, I,NF,0x00014000 STRING "POST CODE (PSP): PSPSTATUS_BOOTLOADER_SUCCESSFULLY_ENTERED_C_MAIN"
INFO STS 0x000057c6[0x00014000] SVC, S, M, I,NF,0x00014000 STRING "POST CODE (PSP): PSPSTATUS_BOOTLOADER_LOADED_SMU_FW_SUCCESSFULLY"
INFO STS 0x000057c6[0x000154b] SVC, S, M, I,NF,0x00014000 STRING "POST CODE (PSP): PSPSTATUS_BOOTLOADER_LOADED_AGESA0_FROM_SPIROM_SUCCESSFULLY"
INFO STS 0x000057c6[0x000154b] SVC, S, M, I,NF,0x00014000 STRING "POST CODE (PSP): PSPSTATUS_MP1_TAKEN_OUT_OF_RESET"
INFO STS 0x000057c6[0x0001665] SVC, S, M, I,NF,0x00014000 STRING "POST CODE (PSP): PSPSTATUS_PSP_AND_SMU_CONFIGURED_WAITED"
PROXY MODE

- Passthrough hardware accesses to real hardware
- Stub running on real PSP
- Reads/Writes to devices get captured by generic proxy component and forwarded to the real hardware
- Which communication channel to use?
PSP STUB TRILOGY: EP 1: A SPI FLASH HOPE

- Use SPI Flash interface and emulator
- Exchange data using SPI Flash Read and Page Program requests
- Works reliable but slow (2-3 accesses per second)
- Requires an expensive flash emulator

https://github.com/PSPReverse/em100/tree/network-mode-v1
EP 2: THE PPR STRIKES BACK

- Found AMD PPR
- Explains Low Level SPI register interface
- We can execute arbitrary commands now!
- Enables use of DediProg EM100 Hyper Terminal
- Blazingly fast (don’t forget to disable Nagle for TCP!)
- Still requires an expensive flash emulator 😞
EP 3: RETURN OF THE UART

- Explore use of the legacy UART for a low cost solution
- SuperIO chip attached to the SoC via LPC
- Need correct sequence to enable UART
- Analyze SuperIO accesses over the LPC bus from logic capture (lpc-dec)
- Much slower than SPI but very cheap

https://github.com/AlexanderEichner/lpc-dec
INSERTING THE PROXY

1. Setup stack
2. Map SPI flash
3. Load and verify AMD public key
4. Load and verify Off-Chip bootloader

The On-Chip BL needs to validate the size of the off-chip BL!
FIRMWARE FILE SYSTEM

File

- Header
- Body
- Signature
  (optional)

Directory

Secondary Directory

Entry
• Offset 0x14: Bodysize (0xc640)
• Offset 0x64: Load address (0x100)
INSERTING THE PROXY

Off-Chip BL is copied into SRAM and then verified (to avoid TOCTOU).

The header is processed before the signature is checked.

-> Input validation is required.
void load_off_chip_bl (void) {
    ...
    if (abs(load_address) + abs(body_size) > ROM_BL_STACK)
        return -1;
    ...
    copy_bl(load_address, spi_src, body_size
    return 0;
}

Validated size: 0xc640
Used size: 0x8000c640
**INSERTING THE PROXY**

1. Place PspStub in SPI flash
   - Appended with return addresses
2. Flip sign-bit of body size
3. Success!

```plaintext
SRAM

PspStub

On-Chip Bootloader

SPI Flash

PspStub

copy_bl(..., ..., 0x8000c640)

On-Chip Bootloader

32 Bit / 4 GB
```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>u32Dw0</td>
<td>0x00500011</td>
</tr>
<tr>
<td>cbSrc</td>
<td>2147534400</td>
</tr>
<tr>
<td>u32AddrSrcLow</td>
<td>0x02149500</td>
</tr>
<tr>
<td>u16AddrSrcHigh</td>
<td>0x00000000</td>
</tr>
<tr>
<td>u16AddrKeyHigh</td>
<td>0x00000000</td>
</tr>
<tr>
<td>u16SrcMemType</td>
<td>0x00000006</td>
</tr>
<tr>
<td>u32AddrDstLow</td>
<td>0x00000100</td>
</tr>
<tr>
<td>u16AddrDstHigh</td>
<td>0x00000000</td>
</tr>
<tr>
<td>u16DstMemType</td>
<td>0x00000002</td>
</tr>
<tr>
<td>u32AddrKeyLow</td>
<td>0x00000000</td>
</tr>
<tr>
<td>u16AddrKeyHigh</td>
<td>0x00000000</td>
</tr>
<tr>
<td>u16KeyMemType</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

**Copy size:** 0x8000c640

**Load address:** 0x100
**Issue Feature summary**

**AFFECTED SUPPORTED SYSTEMS**

- Zen and Zen+ CPUs (probably)
  - Confirmed:
    - Zen: Ryzen 1700X, Epyc 7281
    - Zen+: Ryzen PRO 3500U, Ryzen 5 2600
- Zen2 is NOT affected

**DISCLOSURE TIMELINE**

- Reported to AMD 26th February 2020
- ...
- Response: 11th May 2020!
- Known bug
  - “AMD has developed mitigations in various products where appropriate.”
PSPEMU BASICS

./PSPEmu

--emulation-mode on-chip-bl
--flash-rom uefi.ROM
--on-chip-bl on-chip-bl.bin
--trace-log /tmp/log
--trace-svcs
--dbg <port>

- Sets starting point in boot process
- Flash image to use for emulated SPI flash
- Sets on chip BL binary
- Trace log destination
- Configures syscall tracing
- GDB stub
PSPEMU FEATURES

- Trace all I/O accesses (MMIO, SMN, x86)
- Intercept and trace all syscalls
- GDB stub for debugging (source level debugging for own code)
- Proxy mode for accessing real hardware
- Create coverage traces for later analysis
- I/O record and replay
CURRENT STATE

Working:
- Bootstrap platform when in proxy mode
  - DRAM works!
  - Ryzen 1700X (Zen)
- Stable communication channel with PSP
  - Fast but expensive
  - Slow but very cheap
- Toolchain for writing and debugging your own code
- I/O log record and replay (no access to real hardware required for first steps)
- Basic micropython port for the PSP 😊

Todo:
- Full platform boot (with UEFI)
- Emulate multiple CCDs/PSPs
- Support multiple CCDs/PSPs in the stub
- Investigate SecureOS on Ryzen
- Test Zen+/Zen2 support
- Zen3?
MAY THE CODE BE WITH YOU

- https://github.com/PSPReverse/PSPEmu - Main emulator
- https://github.com/PSPReverse/libpspproxy - PSP proxy base library
- https://github.com/PSPReverse/unicorn - Patched unicorn
- https://github.com/PSPReverse/psp-apps - Contains the PSP stub
- https://github.com/PSPReverse/em100 - For the flash emulator transport channel
- https://github.com/PSPReverse/PSPTool - Analyze UEFI images
- https://github.com/AlexanderEichner/libgdbstub - Generic portable GDB stub library
- https://github.com/AlexanderEichner/micropython - Port of micropython to the PSP