One Glitch to Rule Them All: Fault Injection Attacks against AMD’s Secure Processor

Robert Buhren
Hans Niklas Jacob

Technische Universität Berlin

BLACKHAT EUROPE 2021
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)
Applications

**SECURE ENCRYPTED VIRTUALIZATION (EPYC)**

- **SEV** protects virtual machines in **untrusted** environments by encrypting VM memory
- The AMD SP is responsible for key management
- Paper: “Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation”

**SECURE OS (RYZEN / TR)**

- Firmware **TPM**
- ...

**TRUSTED EXECUTION ENVIRONMENT**

- AMD SP Trusted Execution Environment
- Linux to support **AMD SP TEE API**
FIRMWARE ANALYSIS

Secure Processor is part of AMD CPU.
  • ARMv7-A

Firmware is stored along UEFI FW!

Updatable through UEFI update.
PSPTOOL

Python-based, Decompression, PEM export of keys, Signature update

Command-line interface, Extraction, Signature verification, Duplicate detection

Parsing, Manipulation, Python API, GPLv3

Python-based Command-line interface

Display, extract, and manipulate PSP firmware inside UEFI images

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

```
pip install psptool
```
Signature verification
Decompression
PEM export of keys
Duplicate detection
Python-based
Command-line interface
Python API
Parsing
Extraction
Manipulation
GPLv3
Signature update

https://github.com/PSPReverse/PSPTool

https://media.ccc.de/v/36c3-10942-uncover_understand_own_regaining_control_over_your_amd_cpu
AMD
Secure Encrypted Virtualization (SEV)
“THE CLOUD IS SOMEONE ELSE'S COMPUTER”

Data-At-Rest: disk encryption
Data-In-Transit: e.g. TLS
Data-In-Use: unprotected
Data - At-Rest: disk encryption
Data-In-Transit: e.g. TLS
Data-In-Use: **Memory Encryption** (AES-128)

**SEV: MEMORY ENCRYPTION FOR VIRTUAL MACHINES**

Hypervisor
SEV REMOTE ATTESTATION

SEV’s remote attestation allows a party to validate the authenticity of a remote system.

Customer: Is my VM deployed on a genuine AMD system with SEV protection in place?
Faulting the AMD-SP
FAULT INJECTION ATTACKS

Modifications of an ICs environment can cause errors in the ICs operation

- Lower voltage rails → Voltage fault injection
- Hit IC with electro magnetic radiation → EM fault injection
- Hit IC with laser → Laser fault injection ...
- Most faults are useless for an attacker

input() == “SafePW”

This is the only useful state
Fault Injection Attacks

Modifications of an ICs environment can cause errors in the ICs operation:

- Lower voltage rails → Voltage fault injection
- Hit IC with electromagnetic radiation → EM fault injection
- Hit IC with laser → Laser fault injection ...
- Most faults are useless for an attacker

Key Challenges

- **Trigger:**
  Identify when the IC is in desired starting state

- **Parameters:**
  Which changes to the environment can cause a useful fault

- **Reset/success:**
  Identify failed attacks and retry the attack.

```python
input() == "SafePW"
```

This is the only useful state
AMD-SP BOOT

1. Load & verify AMD_PUBLIC_KEY
   • verify using hash
2. Load & verify PSP_FW_BOOT_LOADER
   • verify using public key
3. Load & verify additional applications
   • verify using public key
1. ROM bootloader

SPI flash

PUBLIC_KEY

Continued SPI activity

AMD_PUBLIC_KEY

PSP_FW_BOOT_LOADER

CS

MISO

PUBLIC_KEY

No SPI activity
CS: Constant # of CS level changes
MISO: CS pulled low

SPI CS: trigger and to determine a successful glitch

Key verification!

CS stays high
ATTACK OVERVIEW

Our goal is to execute our payloads right after the ROM bootloader.

1. Replace AMD_PUBLIC_KEY in UEFI image
2. Replace PSP_FW_BOOT_LOADER component with payload
3. Sign payload with custom key
4. Glitch key verification
DYNAMIC VOLTAGE CONTROL

- SMU monitors SOC and uses the SVI2 bus to communicate with an SOC-external voltage regulator
- SVI2 allows to control two voltage domains per VR
- Ryzen uses single VR, Epyc dedicated VR for each domain
GLITCH SETUP

- Teensy µController to inject packets into the SVI2 bus
- ATX reset line to reset target CPU
- Monitor the SPI bus (CS) to trigger the voltage glitch
- Control glitch parameters via external PC

Glitch steps

<table>
<thead>
<tr>
<th></th>
<th>SVC</th>
<th>SVD</th>
<th>VSoC</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HIGH</td>
<td>HIGH</td>
<td>0.9 V</td>
<td>HIGH</td>
</tr>
<tr>
<td></td>
<td>LOW</td>
<td>LOW</td>
<td>0.8 V</td>
<td>LOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.7 V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.6 V</td>
<td></td>
</tr>
</tbody>
</table>

- SVI2 SVC – clock, CPU/VR (shared)
- SVI2 SVD – data from CPU, pulled low when inactive
- VSoC – target input voltage
- SPI CS – SPI’s chip-select signal (successful/failed pubkey verification)
Glitch steps

- SVI2 SVD: becomes high -> start attack logic
- CPU initially configures voltage
### Glitch steps

- **SVI2 SVD**: becomes high -> start attack logic
- **CPU initially configures voltage**
- **VR constantly sends telemetry data to CPU**
Glitch steps

- SVI2 SVD: becomes high - start attack logic
- CPU initially configures voltage
- VR constantly sends telemetry data to CPU
- Inject packets to disable telemetry - avoids packet collision
• Wait until SPI CS becomes active
Glitch steps

- Wait until SPI CS becomes active
- Count # of CS level changes to time glitch
Glitch steps

- Wait until SPI CS becomes active
- Count # of CS level changes to time glitch
- Inject packet to drop voltage and to revert to the original voltage level
- Verify success by observing CS again -> reset if CS not “low” after timeout
RESULTS

- Epyc and Ryzen CPUs are affected
- Successful glitch between every ~13min (Zen 1) and every ~46min (Zen 3)

Payloads:

- SPI “Hello World”
- Decrypt firmware (Zen 3)
- Dump ROM bootloader to SPI bus
- Deploy custom SEV firmware
- Dump (V)CEK secrets to the SPI bus

https://github.com/PSPReverse/amd-sp-glitch
SEV: AMD-SP
Hosts the SEV firmware that implements the SEV API
Memory encryption keys
Endorsement keys (CEK / VCEK)

Hypervisor

AMD SP
MALICIOUS CLOUD ADMINISTRATOR

• **Debug override**

1. Boot system with patched SEV firmware: Enables the “DBG_Decrypt” SEV API command regardless of a guest’s SEV policy
2. Decrypt the VM’s memory

Works with SEV / SEV-ES and SEV-SNP
SEV REMOTE ATTESTATION

SEV’s remote attestation allows a party to validate the authenticity of a remote system.

Customer: Is my VM deployed on a genuine AMD system with SEV protection in place?
1. AMD-SP creates measurement of SEV protected VM
2. Customer receives signed attestation report including measurement
3. Customer validates attestation report by verifying its signature using a key from an AMD keyserver: (V)CEK
Extracted endorsement keys allow an attacker to, e.g., fake the presence of SEV!
VERSIONED CEK (VCEK) SIMPLIFIED

“[VCEK is] derived from chip-unique secrets and current TCB version”
VCEK ATTACK

What if there is a bug?
OUR ATTACK

• Version is part of the header
• We get VCEK for any TCB
• SEV-SNPs allows TCB downgrade → attack needs only one glitch
Summary

**AMD-SP IS SUSCEPTIBLE TO VOLTAGE FAULT INJECTION ATTACKS**

- **Ryzen** and **Epyc** Zen 1, Zen 2 and Zen 3 systems are affected
  - ThreadRipper most probably
- Allows an attacker to **execute payloads** on the AMD-SP right after the ROM bootloader
- **Reliable code-execution** between every ~13min (Zen 1) and every ~46min (Zen 3)
- **SEV’s** protection mechanism can be circumvented
- **fTPMs** most probably compromised
  - not tested yet
- **Mitigations:** none
  - Future CPU generations might include HW and SW mitigations
RESOURCES

https://arxiv.org/abs/2108.04575
• Paper: One Glitch to Rule Them All: Fault Injection Attacks Against AMD SEV

https://github.com/PSPReverse/amd-sp-glitch
• Supplemental data and code:
  • Glitch setup and code
  • (V)CEK key derivation implementation
  • Firmware decryption implementation

https://github.com/PSPReverse/amd-sev-migration-attack
• Proof-of-concept implementation of the migration attack for SEV / SEV-ES

https://github.com/PSPReverse/PSPTool
• psptool & psptrace

https://github.com/PSPReverse/PSPEmu
• PSPEmulator: Emulator for the AMD-SP
• QEMU port: https://github.com/RobertBuhren/qemu/tree/pspemu
THANK YOU

Robert Buhren: robert.buhren@sect.tu-berlin.de
Hans Niklas Jacob: hnj@sect.tu-berlin.de