DOI:10.1145/3725845

To view the accompanying Technical Perspective, visit doi.acm.org/10.1145/3725863

SRAM Has No Chill: Exploiting Power Domain Separation to Steal On-Chip Secrets

By Jubayer Mahmod and Matthew Hicks

ABSTRACT

The widespread use of embedded systems has increased the risk of physical memory disclosure attacks. A notable example is the *cold boot* attack, where attackers exploit DRAM's temperature-dependent data retention property. At low temperatures, DRAM cells temporarily retain their state after power loss, allowing sensitive data to be recovered. Cold boot attacks can expose system secrets, bypassing defenses like disk encryption. To counter this threat, developers store sensitive data in on-chip SRAM. Unlike DRAM, on-chip SRAM is isolated from external access and, due to its low capacitance, loses data almost immediately when powered off, making it robust against such attacks.

While SRAM protects against traditional cold boot attacks, we show that there is another way to retain information in on-chip SRAM across power cycles. This paper presents *Volt Boot*, an attack that demonstrates a vulnerability of on-chip SRAM due to the physical separation common in modern system-on-chip power distribution networks. *Volt Boot* leverages asymmetrical power states (for example, on vs. off) to force SRAM state retention across power cycles, eliminating the need for traditional cold boot attack enablers, such as low-temperature or intrinsic data retention time. Using three modern ARM Cortex-A SoCs, we demonstrate the effectiveness of the attack in caches, registers, and iRAMs. Unlike other forms of SRAM data retention attacks, *Volt Boot* retrieves data with 100% accuracy—without any complex post-processing.

1. INTRODUCTION

An increasingly connected world makes us dependent on computing devices that handle a wide range of security- and privacy-critical operations. We use smartphones and watches to manage bank transactions and store biometric information. On the industrial and government side, embedded devices monitor remote system operations and feed data critical to industrial processes and national defense. Physical access to these devices leads to a wide range of security exploits, including impersonation, proprietary software cloning, and infiltration and exploitation of industrial and defense infrastructures.

The original version of this paper was published in *Proceedings of the 27th ACM Intern. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2022.

A standard approach to prevent data theft from a system's non-volatile memory (NVM) is to enforce full-disk encryption methods, such as BitLocker and VeraCrypt. These encryption methods protect data using a password or PIN so that even if a device is lost or stolen, NVM remains inaccessible to an attacker. Disk encryption changed the attackers' focus on other types of memories, such as DRAM. Halderman et al. show how an attacker gains access to a disk encryption key by *cold booting* a system and dumping its main memory.[10] In this attack, the authors use low temperature ($-50^{\circ}C$) to 'freeze' the data in DRAM cells so that even if the memory is *out of power* for a short time, it retains its logic states. Once 'frozen,' an attacker physically removes and inserts the victim DRAM in another machine to run forensics on the dumped memory image. While this attack is practical for larger devices where DRAM is removable (for example, laptops), it poses a few technical challenges for mobile and other embedded devices. In an embedded device, memory chips and processors are soldered on the pc boards (PCBs), making it difficult to remove them from a system. FROST[17] overcomes this challenge by allowing device reset to factory default—preserving DRAM's content while the device boots from another media.

To defend devices from cold boot attacks, researchers have proposed numerous methods where sensitive code and data remain encrypted in on-chip memory; decryption occurs only in on-chip memories, such as caches.[16,26] Since systems-on-a-chip (SoCs) already contain large enough on-chip storage to hold keys and cryptographic states, executing software on the memory requires no additional hardware. For example, TRESOR uses x86 debug registers to store sensitive AES states without leaking security-critical information to off-chip memory.[16] Researchers extend this idea to ARM devices, where the CPU fetches the encrypted software and data to the on-chip memories before decryption and execution.[6,7,8,9,23,26] These works advocate fully on-chip execution for cryptographic operations because attacking a processor's internal memory is expensive and demands sophisticated attack methods, such as decapsulation. This article evaluates the security of on-chip computation schemes, specifically under the cold boot attack model, and empirically shows that these methods are secure from the traditional cold boot attack.[10]

On-chip memories, primarily SRAM, are integrated directly into the processor die, offering greater security

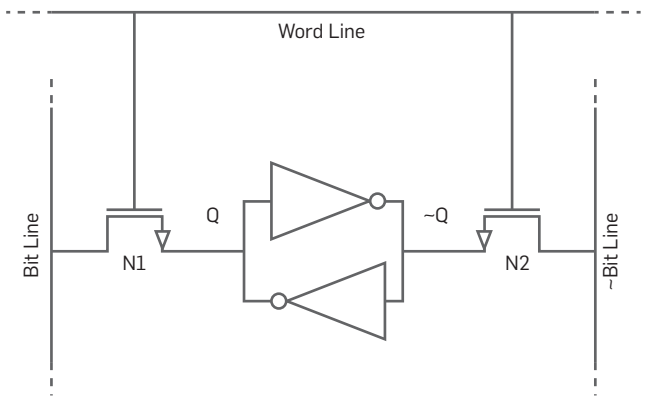
against physical attacks (for example, probing attacks[11,26]) compared to off-chip memories. Research shows that SRAM can partially retain data for a few milliseconds under extremely low temperatures (below $-110^{\circ}C$).[1] However, achieving such temperatures poses significant challenges, risks damaging components such as the battery, and requires specialized equipment. Additionally, exploiting SRAM in embedded devices typically necessitates a physical power disconnect, which far exceeds SRAM's brief data retention time.

This paper introduces *Volt Boot*, a method for executing physical-memory disclosure attacks on on-chip SRAM memories by exploiting SoCs' power domain separation. These domains operate at specific voltages, managed by components such as PMICs, capacitors, and inductors, to balance performance and power efficiency. We demonstrate that traditional cold-boot-style attacks, which leverage low-temperature-induced data retention, are ineffective on embedded SRAM (§Section 3). We uncover a new attack vector that exploits *power domain separation* of modern SoCs and show that such *architectural design choices can be weaponized* for leaking fully on-chip security-critical information(§Section 4). Using *three* commercially available Cortex-A-profile devices, we demonstrate the attack by retrieving data from caches (§Section 6.1), CPU registers (§Section 6.2), and iRAMs (§Section 6.3). Provided that an application runs from internal memory (for example, cache) undisturbed by an operating system's background process, we retrieve memory image with **100% accuracy**. Finally, we analyze the trade-offs involved in potential countermeasures (§Section 7)

2. BACKGROUND

On-chip SRAM. Static random access memory (SRAM) is the building block of temporary on-chip storage, such as caches, iRAM, registers, translation lookaside buffers (TLBs), and branch target buffers (BTBs), making them one of the most common type of memory in modern computing devices. Figure 1 illustrates a typical SRAM cell, which is composed of two inverters in a positive feedback configuration to store one bit of data. Transistors N1 and N2 provide access to the data bit (Q) and its complement (~Q), respectively. Unless a processor executes a read/write command, *Word Line* remains de-asserted with data stored in the cross-coupled structure formed by inverters.

Figure 1. Schematic diagram of a typical SRAM cell.



Unlike other types of memory, *direct access* (that is, direct software read/write) to many types of on-chip SRAM (for example, instruction cache) is uncommon in typical processors. However, most architectures provide access to these internal memories through various methods to debug low-level memory errors; ARM provides co-processor instructions in ARMv8 architecture, while RISC-V processors[22] allow memory-mapped access. For a more concrete example, the Cortex-A72 processor offers access to 15 distinct internal RAMs—including caches, TLBs, and BTBs—via its `cp15` co-processor interface.

Fully on-chip computation. Storing plain-text security-critical information in a DRAM is unsafe, motivating both industry and academic research to push for safer off-chip memory-management schemes. Since Intel's 6th generation, all subsequent processors are able to obfuscate data in the DDR3 and DDR4 DRAMs using session keys and pseudo-random numbers.[15,25] For devices without inline encryption, researchers propose fully SoC-bound computations where sensitive information never leaves a chip's physical boundary.

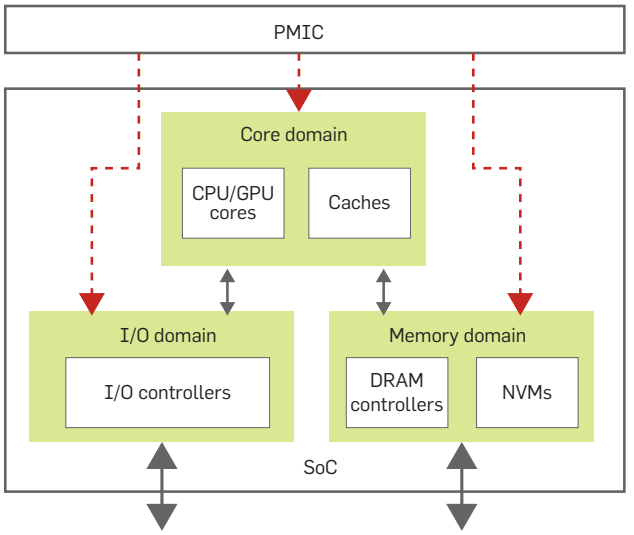
In most cases, it is unnecessary to encrypt and decrypt every memory-bound transaction, as software can store intermediate states in on-chip memories as plain text. This idea inspired numerous on-chip computation methods. Sentry[7] and Copker[9] uses iRAMs and caches as temporary memory to avoid exposing secrets to DRAM. *Cache-assisted secure execution (CaSe)* extends this idea by adding TrustZone support to a partially locked cache. The processor fetches encrypted software from main memory and stores it in a locked cache as plain text. From this point, the plain-text data/code remains in the cache for the duration of execution. Similarly, TRESOR,[16] PRIME,[8] and Security Through Amnesia[23] use on-chip registers (for example, debug) to implement cryptographic algorithms on-chip. These methods essentially emulate a microcontroller's behavior in an application processor and reduce the attack surface to the border of the chip itself. Given no plain-text data is released off-chip, these methods provide strong security against the most sophisticated physical-memory disclosure attacks, for example, cold boot.

Power domain separation. SoCs integrate numerous circuit blocks, each exhibiting unique analog characteristics. To meet stringent performance and power-efficiency requirements, these blocks are divided into separate voltage domains. The power management unit (PMU) within the SoC dynamically manages the voltage levels for these domains at runtime, tailoring them to the workload of each domain.

In modern, complex SoCs, dozens of off-chip supply pins connect to various power domains, enabling precise control over analog circuit behavior. This setup mitigates challenges such as ground bounce, power-supply noise, and per-pin current limitations. We broadly categorize the power-supply domains of an SoC into three main areas, as illustrated in Figure 2:

▸ **Core power domain:** The processing elements are in this domain. For example, the ARM cluster in a multi-core

Figure 2. A simplified block diagram of an SoC's power domains. The PMIC is an external component that maintains a specific voltage level at each power-domain supply pin.



SoC draws power from the core supply-voltage domain. Apart from computing elements such as CPU extensions and GPUs, this domain supplies power to L1 caches and their associated control circuitry.

- **Memory power domain:** This domain supplies power to the memories and their associated peripherals. Most SoCs manage main memory, non-volatile memory (for example, Flash), and L2/L3 caches with this power domain.
- **I/O power domain:** Power for the I/O controllers and external peripherals is drawn from this domain.

SoC designers subdivide power domains into smaller logical blocks that allow fine-grain control of the different components within an SoC. For example, some processors allow powering down individual cache components in its `L1` memory domain through software.[3] The domains are separated using power gating to balance energy consumption and performance of independent blocks at startup and runtime.

3. COLD BOOTING ON-CHIP SRAM IS INEFFECTIVE

In the absence of refresh, a DRAM cell's data-retention time depends on the time it takes for capacitors to leak enough charge to alter the digital value sensed. Temperature affects DRAM cell-discharge rate; lower temperatures increase data-retention time. Reducing the temperature of a memory device keeps the data in the memory for a short period even if the power is turned off, which is the fundamental idea behind the cold boot attack.[10] Both DRAM and SRAM partially retain their data across power cycles for a short time when the temperature is reduced below a certain level[1,2,10] due to their intrinsic capacitance (although DRAM has much longer retention time). We discuss the technical hurdles that prevent a cold boot attack on SRAM as follows:

SRAM's placement. On-chip SRAMs are tightly coupled with processing cores and are built into the processor die itself. Such placement of SRAM makes it inaccessible given the traditional physical attacker threat model. The *embedded nature* of SRAMs complicates the cooling process as well because an attacker needs to freeze the entire device. *Cold booting* this memory requires extreme low temperature and risks bricking of the system.

Short retention time. We cannot launch a cold boot attack on embedded memories by resetting a device through software because a system easily prevents such attempts by purging residual memories as part of a core's power-down sequence.[5] Disconnecting the power from a device is the only reliable way to prevent the system from executing any residual memory-purging routines.

An abrupt power disconnect from a device while executing critical security operations ensures target information remains in SRAM. To power-cycle an embedded device, we must manually disconnect its power supply (for example, the battery). As the literature suggests, SRAM retains information for only a few milliseconds—even under extremely cold conditions—which is insufficient to execute a reboot by manually disconnecting power from a device.[17]

Effect of low temperature. Typically, systems turn off when the operating temperature crosses a certain threshold set by the manufacturers. Executing a cold boot attack on SRAM requires extremely low temperature (below –110°),[1] which is far beyond the operating limit of most devices. We reproduced a similar attack as FROST[17] in cache memory (SRAM, as opposed to the DRAM targeted in the original attack) of a Raspberry Pi 4 (a quad-core cortex-A72 device) to study cold-temperature data retention of its embedded SRAM.[3] We load bare-metal software to populate both the d-cache and i-cache of each core and extract the cached data in a binary image. Then, the device undergoes static cooling in a TestEquity thermal chamber for an hour to stabilize the core temperature. We power cycle the device for a few milliseconds and extract the cache data to compare it to previously stored binaries. Table 1 lists the *mean* mismatch between post-reboot retrieved cache and pre-stored binary for each core at different temperatures. The information retrieval errors indicate almost no data retention even at $-40^{\circ}C$. Note: SRAMs boot up into random states where approximately 50% of the bits are 1s.

Table 1. Errors in d-cache data after a cold boot attack execution in a BCM2711 SoC. We compute a mean error for each core at different temperatures. The fractional Hamming distance between cache content after power cycle and cache's startup state is ~ 0.10, indicating no data retention.

Temperature	0°C	−5°C	−40°C
	Recommended Min.		SoC's hard limit
Error	50.14%	50.06%	50.39%

4. VOLT BOOT

Volt Boot is an attack that exploits an SoC's power-domain separation to induce cross-power-cycle data retention in embedded SRAM. In this section, we show how circuit design choices made for power and performance reasons lead to a new cold-boot-style attack, but with increased precision and fewer requirements than traditional low-temperature-based data-retention attacks.

4.1. Threat model.

Our threat model is based on the cold boot attack,[10] where an attacker has physical access to a device. The most significant modifications to the original cold boot threat model are the location and type of memory. Our target is SRAM embedded within the core of a device, which prevents direct access to the memory contents without damaging the chip. Then the attacker boots the victim device from a media, for example, boot ROM or USB, to dump the uninitialized volatile memory image. This threat model is consistent with the threat model presented in the FROST,[17] Sentry,[7] and CaSE[26] systems. These papers consider an adversary capable of capturing a device (that is, lost or stolen) that is protected against memory disclosure attacks using a lock screen and off-chip memory encryption.

Our threat model extends to headless embedded devices that collect, store, and transfer sensitive information in an unsupervised environment. Note that a number of system-protection methods exist where devices are permanently locked from programming or software updates. The behavior of such devices resembles an application-specific IC (ASIC), and we consider these systems out of the scope of our threat model.

4.2. Inducing data retention.

SoCs need external pins to supply specific voltage to optimize performance and efficiently use energy under fluctuating loads. Usually, a PMIC supplies power to each domain in a particular order to bring up the board from reset. Figure 3 illustrates how a typical PMIC is used to drive an SoC. Generally, LDOs supply power to the domains where voltage fluctuation is limited, whereas domains with high load fluctuation (and dynamic voltage and frequency scaling) use switching regulators to save energy from heat loss. When a power domain of an SoC draws a large current under the demand of software/hardware, the parasitic inductance of the board and the package drop the voltage at the supply lines; this is called droop. To counter droop, the supply voltage pins are extended out of the SoC to connect capacitors so they 'absorb' the current surge, keeping supply voltage closer to nominal. Regardless of the type of regulator used in a power domain, the pins connected to the PMIC require passive components to filter out noise generated during load fluctuation.

Each power domain is power-gated to allow independent control at startup and runtime. Our observation is that if we externally maintain a steady voltage to the pin that supplies power to a target memory domain, it retains SRAM data—even while the rest of the system undergoes a power cycle. We maintain the domain voltage level from an external voltage probe while disconnecting the main supply line to the PMIC. That is, Volt Boot artificially creates SRAM data retention across a power cycle using a voltage probe in a PCB's test pad or a bare passive component's lead connected to a target memory domain.

As discussed in Section 3, cold booting is an impractical way of attacking on-chip SRAM. Even if it is possible to retain information across power cycles, the extracted data is erroneous because SRAM's charge leakage follows a normal distribution, and so some cells will lose their data even for a short power disconnect.[19] Note that SRAM cells are bistable, which makes it harder to look for keys using the algorithm proposed in the original cold boot attack.[10] Since our attack involves no other physical variables (for example, temperature) and memory discharge rate, manufacturing technology node is irrelevant.

4.3. Attack enablers.

While power domain separation serves as the basis for our attack, a combined effect of multiple aspects and attributes of a system enables Volt Boot. We identify each attack enabler as follows:

Ubiquity of SRAM: SRAM is available in every computing device, ranging from resource-constrained microcontrollers to server-class processors. We can induce artificial SRAM state retention in any SoC that has separate SRAM and compute core power domains.

Internal RAMs store data in plain-text: As mentioned in Section 2, cryptography application developers consider on-chip memories safer compared to external memories. As a result, unlike external DRAMs, scrambling or encrypting SRAM's data is uncommon in commodity processors. Thus, access to data residing in on-chip SRAM guarantees a plain-text version of the information, even if external memories implement scrambling or encryption.

Domain-specific exposed power-supply pins: As discussed, performance, energy efficiency, and die area reduction are the primary reasons to expose domain-specific voltage pins out of an SoC; some of these pins are connected to the embedded SRAM. By construction, data remains error-free in SRAM as long as its supply voltage remains above its data-retention voltage. Note, this voltage is well below the nominal supply voltage of the power domain the cells are connected to. The power domain separation through power-gating methods and dedicated pins exposed from the SoCs let us shut down the entire system while the small portion that holds sensitive data 'alive.'

No default RAM reset hardware: SRAM stays at an uninitialized state after booting-up because of two main reasons:

Figure 3. Typical power supply system. The LC combination supply lines are for switching regulators, whereas the decoupling capacitor filters out noise during fluctuating load driven by LDOs.

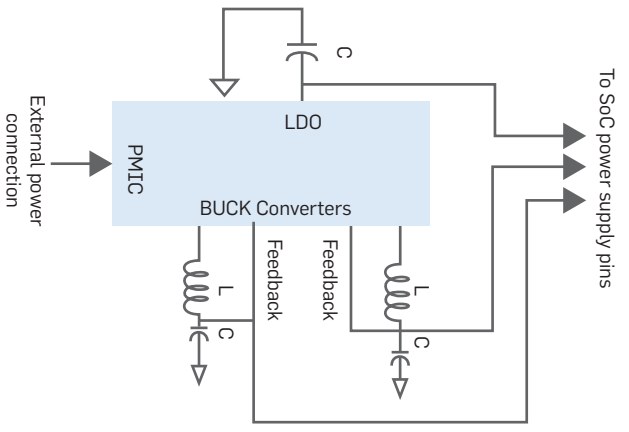
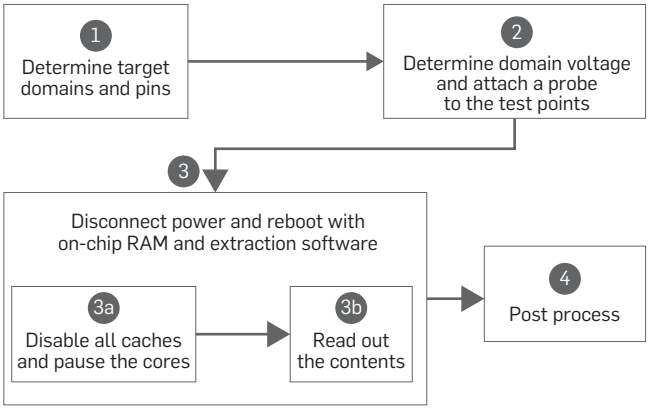


Figure 4. Executing Volt Boot attack.



First, some SRAMs are large (>1MB) compared to the other on-chip resources, and resetting such large memory by iterating over the entire address (usually line by line for caches) space reduces boot speed significantly. Second, SRAM's startup state has numerous security applications, such as PUF[21] and TRNG.[12] Note that cleaning and invalidating a cache at the boot phase does not erase the contents; these operations set the invalid bits to prevent a cache hit, but the data remains unchanged. The co-processor interface still allows reading out the cache contents from a proper exception level (`EL3` for ARM devices). Therefore, initializing cache lines using a software interface is currently the only means to reset the power-on state of L1 caches, which needs the execution of `DC ZVA` instructions for every line.[3,4] The purpose of this instruction is to allow initializing a large block of memory in the cache for a particular data structure without writing zeros in the external memories. Note cleaning/invalidation instructions apply to both instruction and data caches, but *resetting instructions* are exclusive to d-caches.

5. ATTACK EVALUATION

An SoC's domain-separated power-management architecture allows us to supply voltage to a target memory through exposed pins and keep part of the chip active (that is, retaining state) while the rest of the system resets. The actual method used to access embedded SRAM varies depending on the targeted SoC, but devices' power-supply methods are very similar at a high level. We evaluate Volt Boot using ARM devices from different vendors, and to explore the generality of the attack, we choose devices that span a broad range of applications. For example, Raspberry Pis represent a wide array of systems, ranging from embedded devices common in IoT applications to systems capable of running a full-fledged operating system. To expand the targeted memory types, we include an SoC designed for multimedia applications, because it contains large memory-mapped SRAM (that is, iRAM). Table 2 lists the specification of the evaluation platforms. The SoCs in these systems draw power from three different power-management devices, and we observe similar circuit-design choices for the off-chip passive components (see Figure 3).

We use a bench power supply with a current-driving capability of more than 3A to ensure stable voltage delivery to the target power domain of the SoC. Although attaching a probe at the same voltage level typically draws only a few milliamps, a sudden disconnection of the compute core supply line from the PMIC causes a momentary current surge in the target power domain. This surge can lead to a voltage drop below the SRAM's data-retention threshold (§Section 2), resulting in errors in the extracted data. Consequently, a power supply capable of providing sufficient current is crucial, especially when the target memory domain also powers the CPU core(s).

5.1. Attack execution steps.

In this section, we discuss how to execute an attack on SoCs (see Figure 4 for summary).

▸ **Identifying target domains and their associated pins:** To target a device, the first step is identifying the pins supplying power to the SRAM. Directly locating these pins on an SoC is often impractical due to advanced packaging, such as BGA. However, exact pins are not necessary, as supply pins connect to nearby passive components (for example, decoupling capacitors) or test pads, typically near the PMIC. Their layout generally follows a standard pattern (see Figure 3). For our evaluation platforms, Table 3 lists the test points and pin names, with visual representations in Figure 5.

▸ **Attaching a voltage probe:** We measure the nominal voltage at the target pin(s) and connect an external power supply at the same level. The power source must provide sufficient current to maintain the voltage when the device's main power is off, preventing data loss. For example, a Raspberry Pi 4 draws 400–600mA through test pad `TP15` with an 800mV probe, depending on workload. When the main supply is disconnected, the cores draw power from the probe, which keeps the voltage constant even during high momentary current demand. After a few microseconds, current drops to 8mA, and the memory remains in retention state indefinitely.

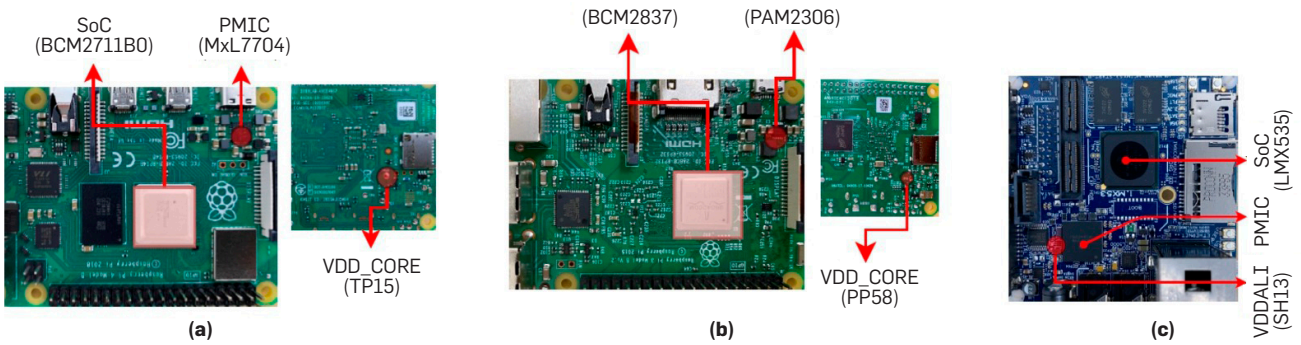
▸ **Power cycling and booting the system:** Once the external probe is in place, we disconnect the device from the main power source while our voltage probe keeps the target SRAM in its data-retention state.

▸ Systems vary in boot-up methods after power down—

Table 2. Evaluation platforms and SoCs.

System on Chip	CPU core	Board	SRAM	Manufacturer
BCM2711	Quad-core Cortex-A72	Raspberry Pi 4 Model B	`L1I: 48KB, L1D: 32KB, L2: 1MB`	Broadcom
BCM2837	Quad-core Cortex-A53	Raspberry Pi 3 Model B	`L1I: 32KB, L1D: 32KB, L2: 512KB`	
i.MX535	Cortex-A8	i.MX53 Evaluation board	`L1I:32KB, L1D: 32KB, L2: 256KB, iRAM: 128KB`	NXP

Figure 5. Pictures of our evaluation platforms (a) Raspberry Pi 4, (b) Raspberry Pi 3, and (c) i.MX535, showing the test points we attach our voltage probe to.



some require user data erasure to boot from alternative media, while others boot internally without external media. We emulate this by booting Raspberry Pis from USB storage and use a post-reboot data-extraction program to:

a. Reduce contamination on the SRAM's retained data during boot-up by avoiding storing data to it (either explicitly or implicitly).
b. Exfiltrate data from the SRAM to other memory (for example, Flash, DRAM, or a debugger) for post-processing.

The cache extraction software uses `CP15` instructions to read cache data into general-purpose registers. For out-of-order processors, data (`DSB SY`) and instruction (`ISB`) synchronization barriers must follow cache access instructions, such as the Cortex-A72's `SYS #0, c15, c4, #0, <xt>` for RAMINDEX operations. Data is then transferred from CPU registers to DRAM via standard load/store instructions.

For iRAM, we dump data directly using the debug interface. On the i.MX535 (Cortex-A8), which boots without external firmware, we connect a `JTAG` probe to automate iRAM reading process.

▸ **Analyzing the memory contents:** The attacker must tailor post-processing strategies based on the target SRAM and their objectives. Volt Boot reads memory without errors, but noise in the extracted data can arise from the dynamic behavior of the software (in cache) during an attack. For instance, extracting cryptographic keys from cache memory with precision depends heavily on the workload of the processing core and the influence of background processes.

5.2. Attacker accessible memory area.

At startup, the CPU uses part of the embedded SRAM before even an attacker has access to those memories. What percentage of memory is available after SoC boot-up depends on the target memory type of an SoC. To find the accessible proportion of the SRAM, we execute bare-metal software that populates a target memory with predefined patterns. The bare-metal setup allows us to calculate the effect of the CPU's boot phase on internal memories, avoiding dynamic behavior, such as cache eviction. Once the software loads the data/instruction in the target memory, we execute the steps discussed above.

Our experiments on the L1 caches of the BCM2711 and BCM2837 show no clobbering during the initial boot phase, consistent with the fact that L1 caches in these SoCs are software-enabled. This allows an attacker to avoid activating the cache, thereby accessing its full contents.

In contrast, these Broadcom devices feature a built-in video core that shares the L2 cache with the ARM CPU cores. During startup, the video core uses pre-compiled binaries for system initialization, overwriting L2 cache contents and preventing any post-reboot data recovery.

The i.MX535 has similar behavior for caches, but boot ROM uses part of the iRAM as scratchpad memory before initializing the DRAM controllers. That is, the CPU resets part of iRAM before allowing any debug connection or software execution. Such a boot method is standard among Cortex-M devices; they clobber 2*KB* SRAM (main memory) at the boot phase.[14,24] Our experiments show that approximately 95% of an i.MX535's iRAM is exposed to Volt Boot.

6. ATTACK EXECUTION IN DIFFERENT SRAM

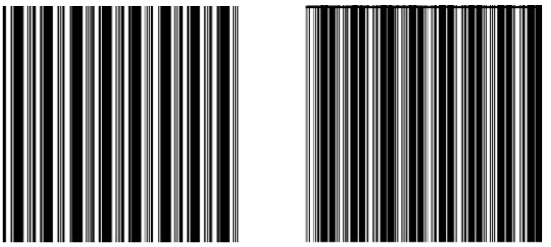
A successful cold boot attack depends on several factors, including temperature and the intrinsic data-retention time of SRAM, which is largely determined by its manufacturing technology. To execute the attack, the attacker must abruptly power off the device to avoid memory corruption or defensive data-wiping mechanisms.

Next, the attacker reduces the device's temperature, as lower temperatures significantly enhance the accuracy of

Table 3. Volt Boot evaluation platforms, test pads, and their nominal voltages.

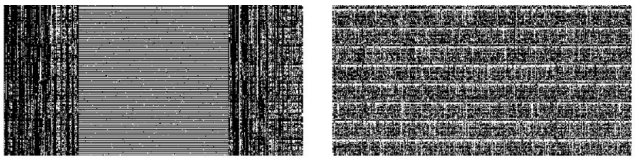
Boards	PCB Test Pads to Probe	Nominal Voltage	Target Memories	Power Domains
Raspberry Pi 3	`PP58`	1.2V	L1D, L1I, registers	Core (VDD_CORE)
Raspberry Pi 4	`TP15`	0.8V	L1D, L1I, registers	Core (VDD_CORE)
i.MX53	`SH13`	1.3V	iRAM	Memory (VDDAL1)

Figure 6. Snapshots of i-cache after attacking bare-metal software in (a) BCM2711 and (b) BCM2837 SoCs. Uninitialized cache cells power on into random state, but when we execute Volt Boot attack, instructions stay in the i-cache across power cycles.



(a) (b)

Figure 7. Snapshots of the caches after executing Volt Boot on a system running a general application. We generate the cache images from one way of each type of cache.



(a) (b)

the extracted data within a fixed period of power loss. Following this, the victim device needs to be rebooted.

In commercial devices, the most common method of abrupt power-off is physically removing the battery or disconnecting the power supply. However, such operations typically take several hundred milliseconds—far exceeding the data-retention limits of SRAM under standard conditions.[1]

Volt Boot is a non-invasive memory-disclosure attack that uses voltage-induced cross-power-cycle data retention in SRAM. The ultimate result of such data retention resembles a cold-boot-style attack with higher accuracy—without exposing a device to very low temperatures. Volt Boot exploits power-domain separation in modern SoCs, eliminating variables such as data-retention time and temperature. We execute three example attacks using the devices listed in Table 2. Our proof-of-concept attacks empirically demonstrate the vulnerability of computation methods that store secrets as plain-text in caches, registers, and iRAMs.

6.1. Attacking caches.

Attacking caches with bare-metal software: In this scenario, we evaluate how Volt Boot targets a device running bare-metal software, emulating typical embedded systems designed for specialized tasks such as monitoring and data collection. We develop a bare-metal program that enables the caches and executes `NOP` instructions across all four cores, allowing us to accurately measure the extent of software extracted by Volt Boot. To maintain precise control over the execution environment, we implement the software in `aarch64` assembly.

We perform the attack on Raspberry Pi devices and compare the cache content to the ground truth machine code. Figure 6 shows the cached content post-attack, with 100% data retention accuracy across all cores of both devices.

Attacking caches with an OS: We demonstrate how Volt Boot extracts data from a user application running on a general-purpose system—the Linux kernel. The application stores a specific pattern (`0xAA`) in a large data structure and reads it back. During execution, we perform the attack steps outlined in Section 5.1 and visualize the post-attack d-cache snapshots in Figure 7. The d-cache correctly contains the expected pattern (that is, `0xAA`). Additionally, we analyze the i-cache and confirm that all instructions for our application are present, matching the ground truth machine code and occupying consecutive address spaces.

To quantify the effect of a cache's dynamic behavior on a Linux-based system, we write a microbenchmark with variable array size; the benchmark loads the array from the Flash to DRAM (and d-cache). We run the benchmark in a standard Raspberry Pi OS running on a Raspberry Pi 4.[20] This SoC has a 32KB two-way set-associative data cache. We vary the number of 8-byte elements in the array by increasing the size of the array in each set of experiments. The size of the array varies from 12.5% (4KB) of the cache size to full-cache size (32KB); by extension, the number of elements in the array varies from 512 to 2,048. We repeat Volt Boot attack on each array size three times and calculate an average number of elements retrieved for each array size (that is, a total of 12 experiments for four different array sizes). We launch one benchmark process per core, which allows us to analyze how L1 cache's (per core) dynamic behavior affects Volt Boot's data retrieval accuracy. At the time of the attack, the victim system concurrently runs four processes in the four different cores of the Cortex-A72 CPU. In each experiment, our post-processing script compares the array elements with the retrieved cache image of each core.

Table 4. Extracted data from d-cache of a BCM2711 SoC using Volt Boot attack. The size of the d-cache in this SoC is 32KB, which is divided into two ways, $W0$ and $W1$.

		$W0$	$W1$	$W0 \cup W1$	% data extracted
4KB	Core 0	373.0	309.0	512.0	100.00%
	Core 1	338.7	341.0	512.0	100.00%
	Core 2	354.7	340.7	512.0	100.00%
	Core 3	363.0	318.0	512.0	100.00%
8KB	Core 0	591.0	633.0	1,024.0	100.00%
	Core 1	580.7	659.7	1,023.7	99.97%
	Core 2	564.7	656.7	1,024.0	100.00%
	Core 3	581.0	656.7	1,024.0	100.00%
16KB	Core 0	1,177.3	1,067.3	2,048.0	100.00%
	Core 1	1,155.0	1,097.3	2,048.0	100.00%
	Core 2	1,179.7	1,084.7	2,045.0	99.85%
	Core 3	1,114.3	1,139.0	2,048.0	100.00%
32KB	Core 0	1,956.7	1,990.7	3,747.3	91.49%
	Core 1	1,980.3	1,970.7	3,753.0	91.63%
	Core 2	1,984.0	1,977.3	3759.3	91.78%
	Core 3	1,878.0	1,815.3	3,509.0	85.67%

Note that other processes (and the kernel) evict cache lines, therefore, an element of the array can be in both ways of the cache in a modified state. We consider an element of the array present in the d-cache only when the entire 8-byte array element is present in the cache. Table 4 lists the results of the L1 data cache data-extraction experiment. A 4KB array contains 512 array elements, and Volt Boot retrieves all the elements. Volt Boot retrieves approximately 90% of the array elements when the array size is close to the cache size. That is, when the data size approaches the total cache size, information retrieval accuracy decreases. The kernel's background processes introduce errors in the data extraction by evicting cache lines when the size of a data structure is comparable to the cache size. Note that in the case of on-chip crypto, which uses cache locking (for example, *CaSE*[26]), Volt Boot retrieves the entire binary of plain-text software since neither the kernel nor other processes can evict secret-holding cache lines.

6.2. Attacking registers.

Modern SoCs contain different types of CPU registers that are not part of a typical boot sequence, for example, ARM cores use vector registers `<v0...v31>` to process SIMD and floating-point instructions. These registers are relatively large (128-bit) and byte-addressable, making them suitable for storing security-sensitive states (for example, key schedules) of cryptographic algorithms, such as AES. Given our threat model, we investigate whether these registers are vulnerable to Volt Boot.

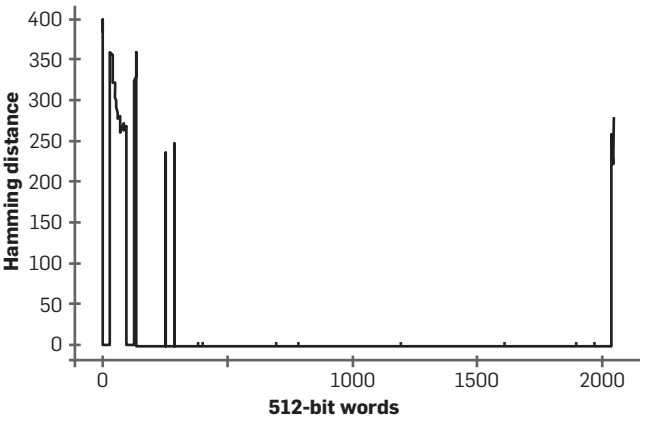
We develop a bare-metal program in `aarch64` assembly that initializes the vector registers with distinct patterns, such as 0xFF and 0xAA. Post-attack analysis on the BCM2711 and BCM2837 reveals that these vector registers completely retain their states during the execution of the Volt Boot attack. Consequently, any on-chip cryptographic program relying on these registers to conceal secrets is vulnerable to Volt Boot.

6.3. Attacking iRAM.

iRAMs (also known as OCRAM) are on-chip memories an SoC uses as temporary storage for different applications, such as boot firmware and multimedia streaming. We study the vulnerability of these memories to Volt Boot attack using a multimedia SoC, the i.MX535,[18] which contains 128KB of iRAM. This memory block is in the L1 power domain, which draws current through `VDDAL1` pin of the SoC. Unlike the BCM2711 and the BCM2837, the i.MX535's ARM core itself draws power through a different pin, `VCCGP`. The i.MX535 boots from internal ROM, and attacking this SoC does not require any external boot media (for example, Flash). That is, this device essentially behaves as a microcontroller at startup. For automation, we attach a JTAG reader to read/write to the iRAM directly and store four copies of a 512×512 (128KB) bitmap image to quantify the accuracy of data extraction through Volt Boot attack.

As detailed in Section 5.2, the entire iRAM cannot be retrieved because the internal boot firmware partially overwrites it before handing control to external software. The overall extraction error is 2.7%. To identify the source, we compute the Hamming distance between the image binary and the extracted iRAM binary at 512-bit granularity (Figure 8). Errors are clustered near the beginning and end of the iRAM, with the largest range between 0xF800083C and 0xF80018CC, consistent with other i.MX535 devices.

Figure 8. Hamming distance between image and post-attack binary.



7. COUNTERMEASURES

To assess potential countermeasures, we break down the *Volt Boot* attack into two broad phases: (a) inducing SRAM data retention across power cycles, and (b) accessing the unmodified SRAM contents after the system reboots. Mitigating the attack requires disrupting at least one of these critical steps. This section outlines potential countermeasures, targeting both phases: the first three approaches address data retention, while the last two focus on preventing access to retained data.

Eliminating power-domain separation. The decision to separate circuit blocks into power domains involves numerous levels of hardware design stack, ranging from device manufacturing to architecture. Eliminating power-domain separation is not a practical countermeasure due to performance, efficiency, and implementation concerns.

Purging residual memory. A straightforward way to avoid on-chip data retention in the caches and other SRAMs is to *erase* the memory as part of the processor's power-down sequence. Such a software/hardware-driven approach is not a practical solution to defend against Volt Boot because an abrupt power disconnect from a live device stops all operations immediately.

Resetting SRAM at startup. Even if an attacker successfully retains the memory states after a power cycle, it becomes useless if there is no feasible method to extract the retained information after a reboot. Resetting the memory using hardware such as *MBIST* prevent this attack.

Our experiments across various devices reveal that hardware SRAM resets during boot are uncommon. Most boot with undefined SRAM states, persisting until overwritten by software. While `armv8.A` allows L2 cache resets via the `nL2RST` pin, this does not apply to L1 caches. Furthermore, the CPU does not benefit from resetting the data and instruction RAMs because the cache operation is dependent

on the status of tag RAMs (for example, L1D-tag and L1I-tag) not the data RAMs (for example, L1D-data and L1I-Data). Data RAMs can only be reset via ISA-provided software *zeroization*, which is not available for all internal RAMs. A simpler alternative is toggling SRAM power at reset, but this hardware-based solution is impractical for existing SoCs due to required hardware modifications.

TrustZone support: ARM TrustZones (TZ) is a hardware-backed memory isolation technique available in most Cortex-A profile processors. Enforcing TZ support prevents unauthorized access (from non-secure state) to memory locations marked secure. An attempt to access a secure line from a non-secure state triggers a hardware exception. The secure data remains inaccessible to an attacker across power cycles when TZ is enforced because the only way to read secure memory content is to change the security attribute of the memory location; such reset erases the memory content.

Mandated authenticated boot: Volt Boot needs to boot the system with an exploitable system image. Signing the system with an OEM's signature and burning the hash of the image in the fuses prevents an attacker from booting a device from another media. Note that all devices do not have mandated authenticated boot functionality as it complicates post-deployment firmware updates for simpler embedded devices. Additionally, some processors boot from an internal boot ROM and use the internal RAM as a scratchpad, providing direct access to on-chip memory. This design is based on the assumption that on-chip SRAM does not retain any data across power cycles.

8. CONCLUSION

The last two decades of hardware security research have seen a rampant increase in proof-of-concept *and* real-world attacks targeting *off-chip memories* designs. Recent efforts to mitigate these attack surfaces primarily turn to *on-chip* computation—that is, *Cache as RAM*—as their deeply embedded nature renders all previous classes of attacks (for example, cold boot) obsolete. However, these systems' reliance on *power-domain separation* enables a new class of attacks: Volt Boot.

In this paper, we show that current on-chip SRAM is indeed resilient against conventional temperature "freezing"-based attacks. However, we show the effectiveness of a voltage-based attack that snapshots SRAM, without exposing an SoC to low temperature, effectively enabling the indefinite retention of SRAM data while software changes. Compared to previous-generation cold-boot attacks against standalone SRAM, Volt Boot achieves *error-free* data exfiltration on devices spanning three distinct microarchitectures—defeating the paradigm of on-chip computation as a viable defense against secret extraction.

9. ACKNOWLEDGMENTS

The project depicted is sponsored by the Defense Advanced Research Projects Agency (DARPA). The content of the information does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

References

1. Anagnostopoulos, N.A. et al. Low-temperature data remanence attacks against intrinsic sram pufs. In *Proceedings of the 2018 21st Euromicro Conf. on Digital System Design*, IEEE, 2018, 581–585.
2. Anagnostopoulos, N.A. et al. Attacking sram pufs using very-low-temperature data remanence. *Microprocessors and Microsystems 71*, 102864, 2019.
3. ARM Limited. ARM Cortex-a72 MPCORE Processor Technical Reference Manual (2016).
4. ARM Limited. ARM Cortex-A53 MPCore Processor Technical Reference Manual (2021).
5. Chan, E.M. et al. BootJacker: Compromising computers using forced restarts. In *Proceedings of the 15th ACM Conf. on Computer and Communications Security*, ACM Press (2008), 555.
6. Chu, D. et al. Ocram-assisted sensitive data protection on arm-based platform. In *Proceedings of the European Symp. on Research in Computer Security*, Springer (2019), 412–438.
7. Colp, P. et al. Protecting data on smartphones and tablets from memory attacks. In *Proceedings of the Intern. Conf. on Architectural Support for Programming Languages and Operating Systems* (2015), 177–189.
8. Garmany, B. and Müller, T. PRIME: Private RSA infrastructure for memory-less encryption. In *Proceedings of the Annual Computer Security Applications Conf.* (2013), 149–158.
9. Guan, L., Lin, J., Luo, B., and Jing, J. Copker: Computing with private keys without RAM. In *Proceedings of the Network and Distributed System Symp.* (2014), 23–26.
10. Halderman, J. A. et al. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM 52*, 5 (2009), 91–98.
11. Henson, M. and Taylor, S. Beyond full disk encryption: Protection on security-enhanced commodity processors. In *Intern. Conf. on Applied Cryptography and Network Security*, Springer (2013), 307–321.
12. Holcomb, D.E., Burleson, W.P., and Fu, K. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Trans. on Computers 58*, 9 (Sept. 2009), 1198–1210.
13. Mahmod, J. and Hicks, M. SRAM has no chill: Exploiting power domain separation to steal on-chip secrets. In *Proceedings of the 27th ACM Intern. Conf. on Architectural Support for Programming Languages and Operating Systems*, Association for Computing Machinery (2022), 1043–1055.
14. Microchip Technology Inc. SAM L10/L11 Family: Ultra Low-Power, 32-bit Cortex-M23 MCUs with TrustZone, Crypto, and Enhanced PTC, (2020).
15. Mosalikanti, P., Mozak, C., and Kurd, N. High performance DDR architecture in Intel® Core™ processors using 32nm CMOS high-k metal-gate process. In *Proceedings of the VLSID Intern. Conf* (2011), 1–4.
16. Müller, T., Freiling, F.C., and Dewald, A. TRESOR runs encryption securely outside RAM. In *Proceedings of the 20th USENIX Conf. on Security* (2011), 17.
17. Müller, T. and Spreitzenbarth, M. Frost: Forensic recovery of scrambled telephones. In *Proceedings of the 11th Intern. Conf. on Applied Cryptography and Network Security*, Springer-Verlag (2013), 373–388.
18. NXP Semiconductors. i.MX535: Multimedia Applications Processors - HD Video, High-Performance, Low-Power, ARM Cortex®-A8 Core (2021).
19. Qin, H. et al. SRAM leakage suppression by minimizing standby supply voltage. In *Proceedings of the Intern. Symp. on Signals, Circuits and Systems.*, IEEE (2004), 55–60.
20. Raspberry Pi *4. Raspberry Pi Foundation* (2021); https://tinyurl.com/yyfmu4rm.
21. Roelke, A. and Stan, M.R. Attacking an SRAM-based PUF through Wearout. In *Proceedings of the IEEE Computer Society Annual Symp. on VLSI*, IEEE (2016), 206–211.
22. SiFive, Inc. SiFive U54-MC manual (2019); https://tinyurl.com/24ee3z5l.
23. Simmons, P. Security through amnesia: a software-based solution to the cold boot attack on disk encryption. In *Proceedings of the 27th Annual Computer Security Applications Conf.*, 2011, 73–82.
24. T. Instruments. SimpleLink ultra-low-power 32-bit ARM Cortex-M4F MCU With Precision ADC, 256KB Flash and 64KB RAM, (2019).
25. Yitbarek, S.F., Aga, M.T., Das, R., and Austin, T. Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors. In *Proceedings of the IEEE Intern. Symp. on High Performance Computer Architecture*, IEEE (2017), 313–324.
26. Zhang, N., Sun, K., Lou, W., and Hou, Y.T. Case: Cache-assisted secure execution on ARM processors. In *Proceedings of the IEEE Symp. on Security and Privacy*, IEEE (2016), 72–90.

Jubayer Mahmod was a research assistant at Virginia Tech, Blacksburg, VA, USA when this paper was written. He is currently a security engineer at Amazon Web Services, Seattle, WA, USA.

Matthew Hicks is an associate professor at Virginia Tech, Blacksburg, VA, USA.