VULNERABILITIES 23 MAR 2022

Another vulnerability in the LPC55S69 ROM

LA Laura Abbott

Here at Oxide, we continue to work on building servers as they should be. Last year, we discovered an undocumented hardware block in the LPC55S69 (our chosen part for our product's Root of Trust implementation) that could be used to violate security boundaries. This issue highlighted the importance of transparency as an Oxide value which is why we are bringing another recently discovered vulnerability to light today. While continuing to develop our product, we discovered a buffer overflow in the ROM of the LPC55S69. This issue exists in the In-System Programming (ISP) code for the signed update mechanism which lives in ROM. This vulnerability allows an attacker to gain non-persistent code execution with a carefully crafted update regardless of whether the update is signed. This can be used to circumvent restrictions when the chip is fully locked down and also extract the device's DICE Unique Device Secret (UDS). Because this issue exists in ROM there is no known workaround other than disabling all hardware and software paths to enter ISP mode. CVE-2022-22819 has been assigned for this vulnerability. Finding two separate issues in the same chip only strengthens Oxide's assertion that keeping code proprietary does not improve product security and hardware manufacturers such as NXP should make their ROM source available for customer review.

Updates are hard

Before discussing the exploit, it's worth thinking about the higher level problem: how do you update your software on a microcontroller once it leaves the factory? This turns out to be a tricky problem where a bug can result in a non-functional device. To make this problem easier, chip makers like NXP will provide some method to put the chip in a mode that allows for safe modification of flash independent of installed firmware. NXP offers this via its In System Programming (ISP) mode.

ISP mode allows a host (typically a general purpose computer) to read and write various parts of the chip including flash by sending commands to the target over a variety of protocols. The LPC55S69 supports receiving ISP commands over UART, SPI, I2C, and, on variants that include the necessary peripheral, CAN. The LPC55S69 can be configured to require code be signed with a specific key. In this configuration, most commands are restricted and changes to the flash can only come via the `receive-sb-file` command.

The update format

The `receive-sb-file` ISP command uses the SB2 format. This format includes a header followed by a series of commands which can modify the flash or start code execution. Confidentiality and integrity of an update are provided by encrypting the commands with a key programmed at manufacturing time, inserting a secure digest of the commands in the update header, and finally signing the header. The C representation of the first part of the header looks like the following:

```c
struct sb2_header_t {
    uint32_t nonce[4];

    uint32_t reserved;
    uint8_t m_signature[4];
    uint8_t m_majorVersion;
    uint8_t m_minorVersion;

    uint16_t m_flags;
    uint32_t m_imageBlocks;
    uint32_t m_firstBootTagBlock;
    section_id_t m_firstBootableSectionID;

    uint32_t m_offsetToCertificateBlockInBytes;

    uint16_t m_headerBlocks;

    uint16_t m_keyBlobBlock;
    uint16_t m_keyBlobBlockCount;
    uint16_t m_maxSectionMacCount;
    uint8_t m_signature2[4];

    uint64_t m_timestamp;
    version_t m_productVersion;
    version_t m_componentVersion;
    uint32_t m_buildNumber;
    uint8_t m_padding1[4];
};
```

The bug

The SB2 update is parsed sequentially in 16-byte blocks. The header identifies some parts of the update by block number (e.g. block 0 is at byte offset 0, block 1 at byte offset 16 etc). The bug comes from improper bounds checking on the block numbers. The SB2 parser in ROM copies the header to a global buffer before checking the signature. Instead of stopping when the size of the header has been copied (a total of 8 blocks or 128 bytes), the parsing code copies up to `m_keyBlobBlock` number of blocks. In a correctly formatted header, `m_keyBlobBlock` will refer to the block number right after the header, but the code does not check the bounds on this. If `m_keyBlobBlock` is set to a much larger number the code will continue copying bytes beyond the end of the global buffer, a classic buffer overflow.

Impact

The full extent of this bug depends on system configuration with code execution possible in many circumstances. A simple version of this can allow for enabling SWD access (normally disabled during ISP mode) via jumping to existing code in ROM. A more sophisticated attack has been demonstrated as a proof-of-concept to provide arbitrary code execution. While code execution via this vulnerability does not directly provide persistence, attack code executes with the privileges of ISP mode and can thus modify flash contents. If this system is configured for secure boot and sealed via the Customer Manufacturing Programming Area (CMPA), modifications of code stored in flash will be detected on subsequent boots. Additionally, ISP mode executes while the DICE UDS (Unique Device Secret) is still accessible allowing for off-device derivation of keys based on the secret.

Mitigation

Because this is an issue in the ROM, the best mitigation without replacing the chip is to prevent access to the vulnerable SB2 parser. Disabling ISP mode and not using flash recovery mode will avoid exposure, although this does mean the chip user must come up with alternate designs for those use cases.

The NXP ROM also provides an API for applying an SB2 update directly from user code. Using this API in any form will still provide a potential path to expose the bug. Checking the signature on an update using another ROM API before calling the update API would provide verification than an update is from a trusted source. This is not the same thing as verifying that the update data is correct or not malicious. Signature verification does provide a potential mechanism for some degree of confidence if using the SB2 update mechanism cannot be avoided.

Conclusion

As exciting as it was to find this issue, it was also surprising given NXP's previous statement that the ROM had been reviewed for vulnerabilities. While no review is guaranteed to find every issue, this issue once again highlights that a single report is no substitute for transparency. Oxide continues to assert that open firmware is necessary for building a more secure system. Transparency in what we are building and how we are building it will allow our customers to make a fully informed choice about what they are buying and how their system will work. We, once again, invite everyone to join us in making open firmware the industry baseline.

Timeline

2021-12-22:
Oxide discovers vulnerability while attempting to understand SB2 update process

2021-12-23:
Oxide discloses vulnerability to NXP

2022-01-03:
NXP PSIRT acknowledges the report

2022-01-04:
NXP PSIRT acknowledges the vulnerability

2022-02-28:
NXP Discloses issues in a NXP Security Bulletin (NDA required) and confirms that a new ROM revision, and thus new part revisions, are required to correct the vulnerability in affected product lines.

2022-03-23:
Oxide discloses as CVE-2022-22819

Servers as they

should be