Shedding too much Light on a Microcontroller’s Firmware Protection

Johannes Obermaier, Stefan Tatschner, August 15, 2017
Shedding too much Light on a Microcontroller’s Firmware Protection

Microcontrollers and Security

The STM32 Security Concept

Attacking the STM32 Security Concept
   Cold-Boot Stepping
   Security Downgrade
   Debug Interface Exploit

Conclusion and Outlook
Microcontrollers and Security

Firmware Protection against Product Piracy

- Microcontrollers in a lot of applications
- Firmware properties
  - Contains intellectual property
  - Might be license-locked
  - Cryptographic keys are included

Gaining access becomes more worthwhile
All firmware contents need to be protected!
Due to insufficient protection, several systems have been broken in the past.
Researchers have shown that security concepts have flaws, hidden functions, or backdoors.

μC
Calibration
Crypt. Keys
Algorithms
Product Piracy
Warranty
Intellectual Property
Licenses
Manipulation
Cloning
Microcontrollers and Security

Firmware Protection against Product Piracy

- Microcontrollers in a lot of applications
- Firmware properties
  - Contains intellectual property
  - Might be license-locked
  - Cryptographic keys are included

⇒ Gaining access becomes more worthwhile
⇒ All firmware contents need to be protected!
Microcontrollers and Security  
Firmware Protection against Product Piracy  

- Microcontrollers in a lot of applications  
- Firmware properties  
  - Contains intellectual property  
  - Might be license-locked  
  - Cryptographic keys are included  

⇒ Gaining access becomes more worthwhile  
⇒ All firmware contents need to be protected!  

- Due to insufficient protection, several systems have been broken in the past.  
- Researchers have shown that security concepts have flaws, hidden functions, or backdoors.
Microcontrollers and Security
The STM32 Series

- STM32: Divided into several families (F0, L0, F1, F2, ...)
- Different capabilities and performance

- **STM32F0**: Entry-level / cost-efficient sub-series
- Used in commercial products
- ARM Cortex-M0 CPU core
- Integrated SRAM, Flash, Peripherals, ...
- No JTAG, only SWD interface for debugging

- Easily available evaluation boards (+integrated debugger)
The STM32 Security Concept
Flash Readout Protection Levels

- Three levels of security available for Readout Protection (RDP)
- Two bytes: nRDP and RDP
- \( nRDP \doteq \sim RDP \) (nRDP is bitwise complement of RDP)
The STM32 Security Concept
Flash Readout Protection Levels

- Three levels of security available for Readout Protection (RDP)
- Two bytes: nRDP and RDP
- \( nRDP \neq \sim RDP \) (nRDP is bitwise complement of RDP)

- **RDP Level 0**: “no protection” (Default)
  Full system access incl. flash read/write
- **RDP Level 1**: “read protection”
  No access to flash memory
- **RDP Level 2**: “no debug”
  SWD interface permanently disabled

<table>
<thead>
<tr>
<th>nRDP</th>
<th>RDP</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55</td>
<td>0xAA</td>
<td>RDP Level 0</td>
</tr>
<tr>
<td>0x33</td>
<td>0xCC</td>
<td>RDP Level 2</td>
</tr>
<tr>
<td>Any other combination</td>
<td>RDP Level 1</td>
<td></td>
</tr>
</tbody>
</table>

Readout Protection Level Configuration
The STM32 Security Concept
Flash Readout Protection Levels

- Three levels of security available for Readout Protection (RDP)
- Two bytes: nRDP and RDP
- \( nRDP \neq \sim RDP \) (nRDP is bitwise complement of RDP)

- **RDP Level 0**: “no protection” (Default)
  Full system access incl. flash read/write
- **RDP Level 1**: “read protection”
  No access to flash memory
- **RDP Level 2**: “no debug”
  SWD interface permanently disabled

But what about SRAM in RDP Level 1?
The STM32 Security Concept
Readout Protection Storage

<table>
<thead>
<tr>
<th>nRDP</th>
<th>RDP</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55</td>
<td>0xAA</td>
<td>RDP Level 0</td>
</tr>
<tr>
<td>Any other combination</td>
<td>RDP Level 1</td>
<td></td>
</tr>
<tr>
<td>0x33</td>
<td>0xCC</td>
<td>RDP Level 2</td>
</tr>
</tbody>
</table>

- RDP and nRDP: Stored in “Option Bytes” region
- Non-volatile memory for system configuration
The STM32 Security Concept
Readout Protection Storage

<table>
<thead>
<tr>
<th>nRDP</th>
<th>RDP</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55</td>
<td>0xAA</td>
<td>RDP Level 0</td>
</tr>
<tr>
<td>Any other combination</td>
<td>RDP Level 1</td>
<td></td>
</tr>
<tr>
<td>0x33</td>
<td>0xCC</td>
<td>RDP Level 2</td>
</tr>
</tbody>
</table>

- RDP and nRDP: Stored in “Option Bytes” region
- Non-volatile memory for system configuration
- Option Bytes: Part of the flash memory
- Flash memory: Part of the system’s memory map
The STM32 Security Concept
Readout Protection Storage

<table>
<thead>
<tr>
<th>nRDP</th>
<th>RDP</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55</td>
<td>0xAA</td>
<td>RDP Level 0</td>
</tr>
<tr>
<td>Any other combination</td>
<td></td>
<td>RDP Level 1</td>
</tr>
<tr>
<td>0x33</td>
<td>0xCC</td>
<td>RDP Level 2</td>
</tr>
</tbody>
</table>

- RDP and nRDP: Stored in “Option Bytes” region
- Non-volatile memory for system configuration
- Option Bytes: Part of the flash memory
- Flash memory: Part of the system’s memory map

⇒ Security impact of flash data manipulation?
The STM32 Security Concept

Flash Protection Logic

- Complex system architecture
- Core and SWD use the same bus for flash access

STM32F0 system architecture
Adapted from: STM32F051 Reference Manual (RM0091)
The STM32 Security Concept

Flash Protection Logic

- Complex system architecture
- Core and SWD use the same bus for flash access
- RDP Level 1 raises special interest: SWD active, but no flash access
- Very little information on flash locking mechanism
  - How does it work?
  - When is the protection triggered?
  - Who manages the protection?

STM32F0 system architecture

Adapted from: STM32F051 Reference Manual (RM0091)
The STM32 Security Concept

Flash Protection Logic

- Complex system architecture
- Core and SWD use the same bus for flash access
- RDP Level 1 raises special interest: SWD active, but no flash access
- Very little information on flash locking mechanism
  - How does it work?
  - When is the protection triggered?
  - Who manages the protection?

⇒ Locking mechanism requires deep investigation and reverse engineering!
Attacking the STM32 Security Concept

Microcontrollers and Security

The STM32 Security Concept

Attacking the STM32 Security Concept
   Cold-Boot Stepping
   Security Downgrade
   Debug Interface Exploit

Conclusion and Outlook
Attacking the STM32 Security Concept

Methodology

- Theoretical analysis of each security concept component
- Discovery of weaknesses, Proof-of-Concept for vulnerability
- Discussion of countermeasures

⇒ **Goal:** Extraction of flash memory contents
Attacking the STM32 Security Concept

Methodology

- Theoretical analysis of each security concept component
- Discovery of weaknesses, Proof-of-Concept for vulnerability
- Discussion of countermeasures

Goal: Extraction of flash memory contents

Three tasks for security testing

1. **Cold Boot Stepping**: Access permissions to non-flash memory / SRAM in RDP Level 1
2. **Security Downgrade**: Feasibility and effects of flash data manipulation
3. **Debug Interface Exploit**: Detailed investigation of flash locking mechanism
Attacking the STM32 Security Concept

Cold-Boot Stepping

- RDP Level 1 often in use
  - On-field debugging
  - Possibility of failed-device analysis
  - OpenOCD support only for RDP Level 0+1

- Access permissions to
  non-flash memory / SRAM in RDP Level 1
Attacking the STM32 Security Concept
Cold-Boot Stepping

- RDP Level 1 often in use
  - On-field debugging
  - Possibility of failed-device analysis
  - OpenOCD support only for RDP Level 0+1

- Access permissions to non-flash memory / SRAM in RDP Level 1

- Microcontroller halted upon connecting a debugger

- Access to SRAM and peripherals allowed!

- Potential weakness!

---

Shedding too much Light on a Microcontroller’s Firmware Protection | Johannes Obermaier, Stefan Tatschner | August 15, 2017 | 9

© Fraunhofer
Attacking the STM32 Security Concept

Cold-Boot Stepping

- Common bootloader implementation: Application CRC validation during startup
- Intermediate results in SRAM, Bytewise-CRC reversible $\Rightarrow$ CRC source data extraction!
Attacking the STM32 Security Concept
Cold-Boot Stepping

- Common bootloader implementation: Application CRC validation during startup
- Intermediate results in SRAM, Bytewise-CRC reversible \( \Rightarrow \) CRC source data extraction!

- Each CRC iteration takes \( T \) microseconds
- Start with \( n = 0 \)

1. Reset System: Set a well-defined initial state
2. Run System for \( n \cdot T \): Allow computation up to the desired intermediate CRC
3. Dump Memory: Read the intermediate CRC from SRAM, compute firmware byte
4. \( n = n + 1 \): Repeat for next firmware byte
Attacking the STM32 Security Concept
Proof of Concept

- Similar to a real (successful) penetration test
Attacking the STM32 Security Concept
Proof of Concept

- Similar to a real (successful) penetration test
- Fully automated attack setup
- Device under Attack: Bootloader computing a CRC32
- Attack control board: Precise Exec.-Time Control
- Power Relay: Reset / Power cycle after each iteration
Attacking the STM32 Security Concept
Proof of Concept

- Similar to a real (successful) penetration test
- Fully automatized attack setup
- Device under Attack: Bootloader computing a CRC32
- Attack control board: Precise Exec.-Time Control
- Power Relay: Reset / Power cycle after each iteration
- On-Line CRC reversing, dynamic timing adjustment
- Extraction of seven bytes per minute

⇒ **Firmware extraction feasible**, but slow
⇒ RDP Level 1 unable to protect firmware
Attacking the STM32 Security Concept
Countermeasures against Cold-Boot Stepping

■ Technical solution
  ■ Do not use RDP Level 1, use RDP Level 2 instead
  ■ Read the datasheet thoroughly (SRAM protection not claimed!)

■ Mitigation / Increasing attack effort
  ■ Insert random delay / timing jitter
  ■ Move computations into CPU registers (weak, attack can be adapted)

■ Increase Discoverability / Awareness, RDP Level 2 support
  ■ Created OpenOCD Patch “Added RDP Level 2 support”
    http://openocd.zylin.com/4111
Attacking the STM32 Security Concept

Security Downgrade

- 16 bits to store RDP Level (3 possible configurations)
- In theory, high redundancy possible

- Hamming-Distance Level 2 to 1: One single bit!
- Flipping any bit causes security downgrade!
- Includes non-complementary bytes
- Dangerous fallback!
Attacking the STM32 Security Concept

Security Downgrade

- 16 bits to store RDP Level (3 possible configurations)
- In theory, high redundancy possible
- But: Non-optimal security design
- 1 setting each maps to RDP Level 0 and 2
- 65534 settings map to RDP Level 1

<table>
<thead>
<tr>
<th>HEX</th>
<th>BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRDP</td>
<td>RDP nRDP</td>
</tr>
<tr>
<td>00</td>
<td>0000 0000</td>
</tr>
<tr>
<td>00</td>
<td>0000 0001</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>0011 0011</td>
</tr>
<tr>
<td>33</td>
<td>0011 0011</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>0101 0101</td>
</tr>
<tr>
<td>55</td>
<td>0101 0101</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>1111 1111</td>
</tr>
<tr>
<td>FF</td>
<td>1111 1111</td>
</tr>
</tbody>
</table>

Level 0

Level 1

Level 2

Flash Readout Protection
Attacking the STM32 Security Concept

Security Downgrade

- 16 bits to store RDP Level (3 possible configurations)
- In theory, high redundancy possible
- But: Non-optimal security design
- 1 setting each maps to RDP Level 0 and 2
- 65534 settings map to RDP Level 1
- Hamming-Distance Level 2 to 1: **One single bit!**
- **Flipping any bit causes security downgrade!**
- Includes non-complementary bytes
- Dangerous fallback!

<table>
<thead>
<tr>
<th>HEX</th>
<th>BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>00</td>
<td>01 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>33</td>
<td>CB 01 01 11 10 10 11 11</td>
</tr>
<tr>
<td>33</td>
<td>CC 01 01 11 10 11 11 11</td>
</tr>
<tr>
<td>33</td>
<td>CD 01 01 11 11 11 11 10</td>
</tr>
<tr>
<td>55</td>
<td>A9 01 01 10 10 10 10 10</td>
</tr>
<tr>
<td>55</td>
<td>AA 01 01 10 10 10 10 10</td>
</tr>
<tr>
<td>FF</td>
<td>FE 11 11 11 11 11 11 10</td>
</tr>
<tr>
<td>FF</td>
<td>FF 11 11 11 11 11 11 11</td>
</tr>
</tbody>
</table>
Attacking the STM32 Security Concept
Reverse-Engineering the Flash Memory Layout

- UV-C light (254 nm wavelength) erases flash memory cells (0→1)
- Die access required → Acid decapsulation

Shedding too much Light on a Microcontroller’s Firmware Protection | Johannes Obermaier, Stefan Tatschner | August 15, 2017 | 14
Attacking the STM32 Security Concept
Reverse-Engineering the Flash Memory Layout

- UV-C light (254 nm wavelength) erases flash memory cells (0→1)
- Die access required → Acid decapsulation

- Experiment: Full-Chip UV-C illumination
- Successful downgrade from RDP Level 2 to 1
- Causes Firmware destruction → not useful
Attacking the STM32 Security Concept
Reverse-Engineering the Flash Memory Layout

- UV-C light (254 nm wavelength) erases flash memory cells (0→1)
- Die access required → Acid decapsulation

- Experiment: Full-Chip UV-C illumination
- Successful downgrade from RDP Level 2 to 1
- Causes Firmware destruction → not useful

- Location of nRDP and RDP bytes unknown
- Masking not possible, yet
- Reverse-Engineering of Flash-Memory Layout
Attacking the STM32 Security Concept
Reverse-Engineering the Flash-Memory Layout + PoC

- Bisection method: Repeatedly cover a part of the flash
- Create simple mask (e.g., piece of plastic)
- Apply UV-C light, analyze flipped bits
Attacking the STM32 Security Concept
Reverse-Engineering the Flash-Memory Layout + PoC

- Bisection method: Repeatedly cover a part of the flash
- Create simple mask (e.g., piece of plastic)
- Apply UV-C light, analyze flipped bits

- Firmware Flash Layout: 1024 bitlines, 512 wordlines
- nRDP + RDP in lower region

Bisection step, Blue dot = bitflip
(Upper half covered)
Attacking the STM32 Security Concept
Reverse-Engineering the Flash-Memory Layout + PoC

- Bisection method: Repeatedly cover a part of the flash
- Create simple mask (e.g., piece of plastic)
- Apply UV-C light, analyze flipped bits

- Firmware Flash Layout: 1024 bitlines, 512 wordlines
- nRDP + RDP in lower region

- Cover flash except nRDP + RDP
- Very few firmware errors down to no errors

⇒ RDP Level 2 to 1 Security Downgrade possible!
Weak RDP level design!
Attacking the STM32 Security Concept
Countermeasures against Security Downgrade

- Root-cause not fixable by user
  - Non-optimal protection level design
  - RDP Level 2 still recommended, raises the bar for the attacker

- Mitigation available
  - Check for RDP Level 2 during boot process
  - Stop firmware execution if not RDP Level 2, rewrite configuration
  - Prevents Cold-Boot Stepping after security downgrade
  - Negligible performance+memory overhead
Attacking the STM32 Security Concept
Debug Interface Exploit

- Goal: Analysis of the flash protection mechanism
- SWD access to flash prevented in RDP Level 1
- ST-LINK debugger triggers protection instantly

⇒ Implement own SWD debugger
- Less aggressive SWD interface initialization
- Only a (bus) access triggers flash lockdown!

Digging deeper into the system...

Anomaly: If the first bus access targets flash memory, valid data is sometimes returned!

Flash Lock mechanism fails!

Integrated ST-Link on Eval Board

Independent ST-LINK (clone)
Attacking the STM32 Security Concept

Debug Interface Exploit

- Goal: Analysis of the flash protection mechanism
- SWD access to flash prevented in RDP Level 1
- ST-LINK debugger triggers protection instantly

⇒ Implement own SWD debugger
- Less aggressive SWD interface initialization
- Only a (bus) access triggers flash lockdown!
- Digging deeper into the system...
Attacking the STM32 Security Concept
Debug Interface Exploit

- Goal: Analysis of the flash protection mechanism
- SWD access to flash prevented in RDP Level 1
- ST-LINK debugger triggers protection instantly

⇒ Implement own SWD debugger
- Less aggressive SWD interface initialization
- Only a (bus) access triggers flash lockdown!
- Digging deeper into the system...

- Anomaly: If the first bus access targets flash memory, valid data is sometimes returned!
- Flash Lock mechanism fails!
Attacking the STM32 Security Concept
Searching for the Root-Cause

- Issue not visible to ST-LINK debugger
  - Very verbose SWD initialization
  - Reading of system config, breakpoints, etc.
  - Flash lockdown triggered early

- Flash locking handled by flash module
- Success ratio: Dependant on bus load
- Instant bus arbitration required
- Race condition! Access vs. flash lockdown
- Lockdown signal arrives a few cycles too late

Shedding too much Light on a Microcontroller's Firmware Protection | Johannes Obermaier, Stefan Tatschner | August 15, 2017 | 18
Attacking the STM32 Security Concept

Using the Exploit

- Exploitable for firmware extraction

1. Apply power cycle for reset
2. Enable debug interface (minimum initialization)
3. Set AHB Access Port to 32 bit width (optional)
4. Trigger AHB Read from desired flash address
5. Receive extracted data
6. On success: Continue with address+4

[...] SWD RESET
[!] Triggered AHB Read at 0x00000100 [OK]
Read from 0x00000100: 0x12345678 [OK]
SWD RESET
[!] Triggered AHB Read at 0x00000104 [OK]
Read from 0x00000104: 0xFFFFFFFF [ERROR]
SWD RESET
[!] Triggered AHB Read at 0x00000104 [OK]
Read from 0x00000104: 0x0800E125 [OK]
SWD RESET
[!] Triggered AHB Read at 0x00000108 [OK]
Read from 0x00000108: 0x2000014A [OK]
SWD RESET
[!] Triggered AHB Read at 0x0000010C [OK]
Read from 0x0000010C: 0x200002A0 [OK]
[...]

Shedding too much Light on a Microcontroller’s Firmware Protection | Johannes Obermaier, Stefan Tatschner | August 15, 2017 | 19

© Fraunhofer
Attacking the STM32 Security Concept
Using the Exploit

■ Exploitable for firmware extraction

1. Apply power cycle for reset
2. Enable debug interface (minimum initialization)
3. Set AHB Access Port to 32 bit width (optional)
4. Trigger AHB Read from desired flash address
5. Receive extracted data
6. On success: Continue with address+4

■ Access may fail ⇒ Retry
■ Readout at 45 bytes per second
■ Practically feasible!

[...]
SWD RESET
[!] Triggered AHB Read at 0x00000100 [OK]
Read from 0x00000100: 0x12345678 [OK]
SWD RESET
[!] Triggered AHB Read at 0x00000104 [OK]
Read from 0x00000104: 0xFFFFFFFF [ERROR]
SWD RESET
[!] Triggered AHB Read at 0x00000104 [OK]
Read from 0x00000104: 0x0800E125 [OK]
SWD RESET
[!] Triggered AHB Read at 0x00000108 [OK]
Read from 0x00000108: 0x2000014A [OK]
SWD RESET
[!] Triggered AHB Read at 0x0000010C [OK]
Read from 0x0000010C: 0x200002A0 [OK]
[...]
Attacking the STM32 Security Concept

Proof of Concept

WOOT'17: Shedding too much Light on a Microcontroller's Firmware Protection

STM32F0 Debug Interface Exploit Demo

Johannes Obermaier, Stefan Tatschner
2017 Fraunhofer Institute AISEC
Attacking the STM32 Security Concept
Impact and Countermeasures

- RDP Level 1 security successfully leveraged!
- Affects STM32F0 only (no success for other series)
- Dangerous for system security
  - Combination of security downgrade + firmware extractor
  - Integrity of flash after downgrade not required anymore
  - Pulls down the requirements on an attacker
- Recommendation: Never use RDP Level 1 → use Level 2
- Requires the attacker to open the device
- Hope for a new hardware revision and fix
Conclusion and Outlook

- Discovery of three major security issues in the STM32F0 series
- Demonstration of their practical relevance
- Presentation of countermeasures and limitations
- Further investigation necessary (other series, etc.)
- Weaknesses perhaps already known to professional adversaries...
Availability

Supplemental materials include scripts, sources, and ELF files for:

- The device under attack (Sample data + CRC implementation)
- The timing control board (Cold-Boot Stepping)
- The Firmware Extractor (Debug Interface Exploit)
- The PoC Video for Firmware Extraction (firmware-extraction.mp4)

Available under the MIT license at
https://science.obermaier-johannes.de/