Fault Injection on Diagnostic Protocols

Niek Timmers
Principal Security Analyst, Riscure
timmers@riscure.com / @tieknimmers
Big shout out to Ramiro and Santiago!

• Security Analysts at Riscure’s **Automotive Security Team**
• Riscure
  • **Services**
    • E.g. Penetration Testing, Security Architecture Review, and more.
  • **Tools**
    • E.g. Automotive Security Test Tools, Fault Injection, and more.
  • **Training**
    • E.g. Embedded Security for Automotive, Fault Injection, and more.
• Offices in the Netherlands, USA and China

*Combining services and tools for fun and profit!*
The hacker’s approach for hacking ECUs

Access to the firmware is a tremendous convenience for an attacker!
Obtaining ECU Firmware

• Firmware is available through official channels
• Firmware is stored in an external memory chip
• Firmware upgrades are not encrypted
• Firmware is leaked / distributed illegally
• Code protection features are not enabled
• Firmware is extracted using a software-based attack

What if all of the above is not applicable? Attackers will resort to something else...
Others came to similar conclusions...

Reference: https://derrekr.github.io/3ds/33c3/#/18

Hackers nowadays use Fault Injection!
Fault Injection – Introduction

“Introducing faults in a target to alter its intended behavior.”

```java
... 
if( key_is_correct ) <-- Glitch here! 
{
    open_door();
}
else 
{
    keep_door_closed();
}
...
```

How can we introduce these faults?
Fault Injection – Techniques

- Clock
- Voltage
- Electromagnetic
- Laser

- A controlled environmental change leads to altered behavior in the target
- They leverage a vulnerability in hardware
Fault Injection – Why does it work?
Fault Injection – Basic concept

Glitch!
Fault Injection – Typical faults

• **Instruction corruption**
  • Executing different instructions
  • Skipping instructions

• **Data corruption**
  • Reading different data
  • Writing different data

*These faults change the intended behavior of software!*
Fault Injection – Tooling

Open source

ChipWhisperer®

Commercial

Inspector Fault Injection

Fault Injection tooling is available to the masses!
Fault Injection – Examples

Bypassing Secure Boot using Fault Injection

Niek Timmers
timmers@riscure.com
(@tieknimmers)

Albert Spruyt
spruyt@riscure.com

November 4, 2016
Fault Injection – Examples

Defeating Secure Boot with EMFI

Ang Cui, PhD & Rick Housley
{a|r}@redballoonsecurity.com
Fault Injection – Examples

KERNELFAULT:
R00ting the Unexploitable using Hardware Fault Injection

Niek Timmers
Senior Security Analyst
@tieknimmers / niek@riscure.com

Cristofaro Mune
Product Security Consultant
@pulsoid / c.mune@pulse-sec.com
We can inject faults and alter software...

What software?
UDS!

It’s common and includes convenient functionality!
UDS – Unified Diagnostic Services

- Diagnostic and Communication Management
  - Diagnostic Session Control
  - Security Access
- Data Transmission
  - Read and Write memory
- Upload / Download
  - Read and Program flash
- Manufacturer proprietary

Disclaimer: Manufacturers are free to implement only a subset of the UDS specification!
UDS – What speaks it?

Most ECUs in a modern car speak it using a multitude of protocols

CAN  K-Line  FlexRay  Ethernet

It is unlikely UDS will go away!
UDS – Typical use case

*Firmware update*

1. Session Control
2. Security Access
3. Request Download
4. Transfer Data
5. ECU Reset

update()
Standard Security Access Check

The secret used by the key calculation algorithm should be protected!
int key_verification(...) {
...
// key verification
if ( received_key == calculated_key ) {
    access_granted();
    error_code = NONE;
} else {
    error_code = INVALID_KEY;
}
return error_code;
}
Standard Security Access Check

• Not successful :'(

• There’s a 10 minute timeout after 3 failed attempts

• Simply not practical for us (or an attacker)

Some times you have to take your losses and move on!
Reading Memory

No restrictions on failed attempts!
int read_memory_by_address(...) {
    ...
    // check if authenticated
    if ( authenticated ) {
        error_code = NONE;
        memcpy(buffer, address, length);
    } else {
        error_code = SECURITY_ACCESS_DENIED;
    }
    return error_code;
}
Reading Memory

- Successful on multiple instrument clusters

- Depending on the target
  - Allows reading out \( N \) bytes from an \textit{arbitrary} address
  - Extraction of the internal memories in \( N \) days

- Addresses include volatile and non-volatile memories
  - Complete firmware extracted

\textit{Extraction of the firmware only has to be done once!}
We have the firmware... now what?
Lots of “cool” stuff...

- Reverse engineering
  - Understanding the device
  - Extracting secrets
- Finding vulnerabilities

*Please see Alyssa’s presentation on reverse engineering firmware efficiently!*
But... does this scale!?
Hardware attacks scale!

• Firmware can be distributed
• Secrets can be distributed
• Vulnerabilities (and exploits) can be distributed
• Vulnerabilities can be exploited remotely
Reading memory is fun!
What about something cooler?
Attack #3: Writing memory

Where should we write to get code execution on the ECU?
Wrapping up...
Why is UDS vulnerable?

• A robust Security Access check is not part of the standard

• Typical Security Access check based on pre-shared secrets

• No fault injection resistant hardware used in most ECUs

• No fault injection resistant software used in most ECUs

What can you do?
Improving Products

• Include fault injection attacks in your threat model

• Design and implement fault injection resistant hardware
  • Start from an early design
  • Test, test... and test again!

• Implement fault injection resistant software

• Make critical assets inaccessible to software
  • E.g. Using “real” hardware
Fault Injection Hardened Firmware

Not hardened

```c
if ( authenticated ) {
  error_code = NONE;
  something_useful();
}
```

Hardened

```c
if ( authenticated ) {
  error_code = NONE;
} else { return }

if ( authenticated ) {
  something_useful();
} else { return }
```

Prevent single point of failures for security critical checks! More info here.
Key takeaways

• Fault injection attacks are available to the masses

• Fault injection attacks subvert software security models

• All unprotected devices are vulnerable

• Presented attack not unique; most ECUs affected

• Fault injection attacks result in scalable attacks
Thank you!