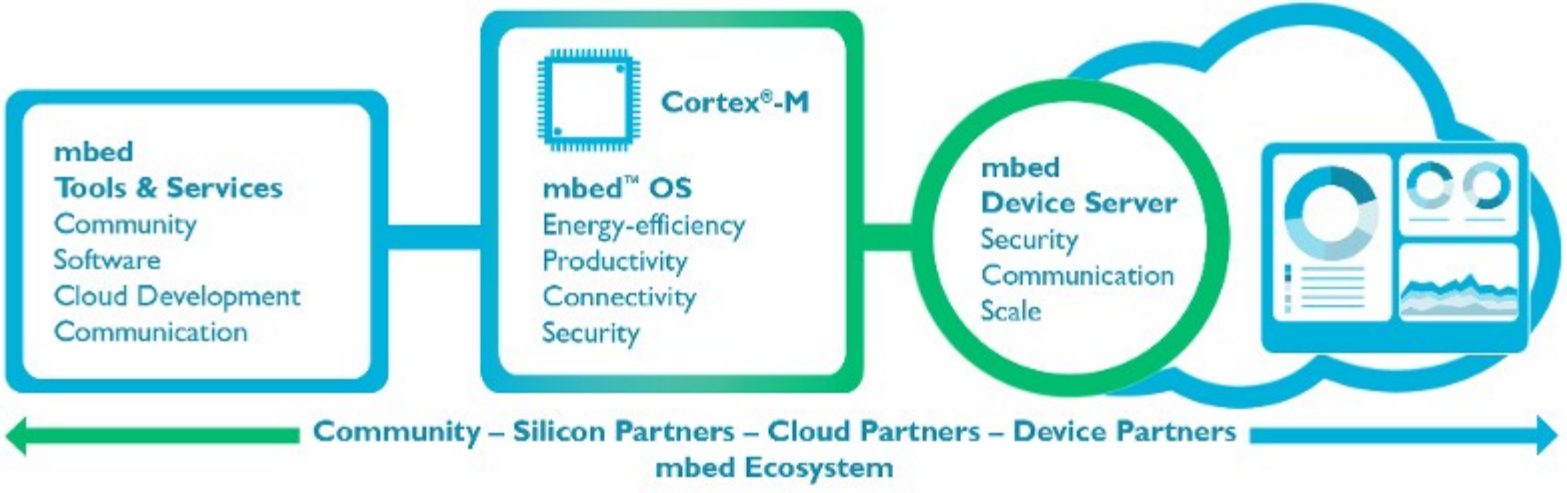
uVisor Debugging Facility Improvements for ARM mbed

Jim Huang (黃敬群) <jserv>
張家榮 <JaredCJR>

OpenIoT Summit / Apr 6, 2016

About the presentation

- IoT systems require effective security frameworks
- advanced security features in mbed OS for ARM Cortex-M processor
  - **uVisor** is a tiny hypervisor / microkernel-like security kernel at the foundation of mbed OS
  - memory protection unit (MPU) is used to compartmentalize code and sensitive data
- Debugging facility for uVisor / ARM mbed

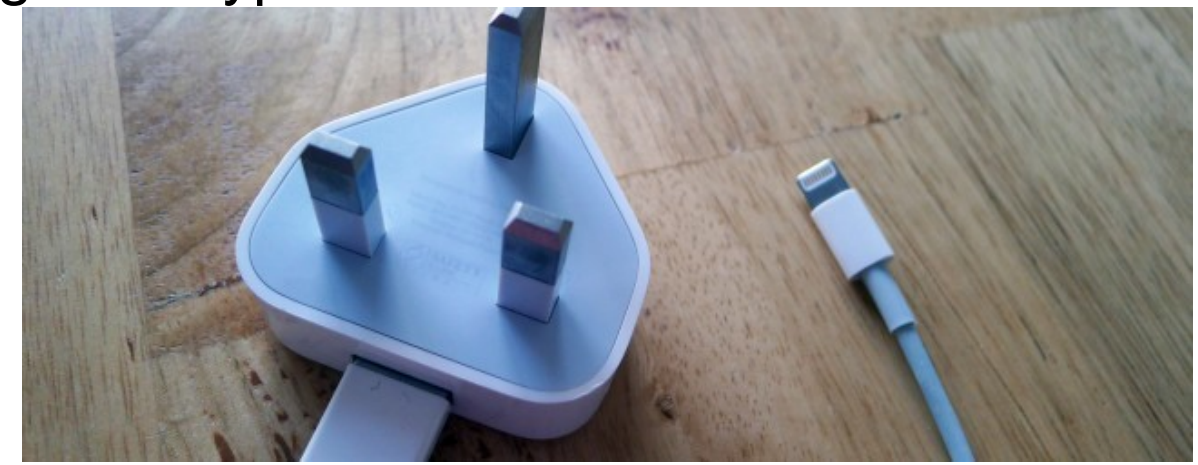


"Security is not
a product, but a process"
- Bruce Schneier

(American cryptographer, computer security and privacy specialist)

Case Study: Attack iOS through USB charger!

- BlackHat 2013
  - MACTANS: INJECTING MALWARE INTO IOS DEVICES VIA MALICIOUS CHARGERS
- "we demonstrate how an iOS device can be compromised within one minute of being plugged into a malicious charger. We first examine Apple's existing security mechanisms to protect against arbitrary software installation, then describe how USB capabilities can be leveraged to bypass these defense mechanisms."



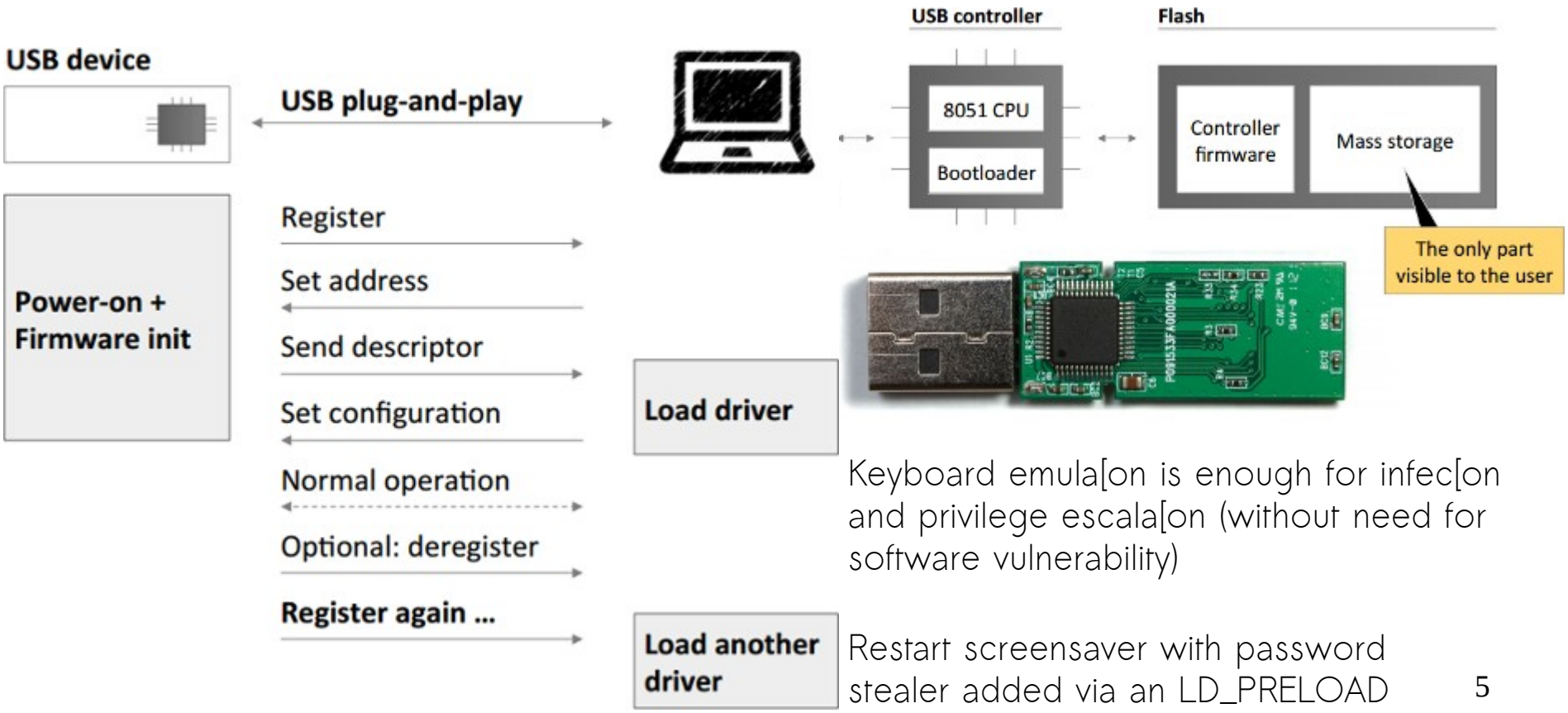
Source:http://www.blackhat.com/us-13/briefings.html#Lau

Case Study: BadUSB

- BlackHat 2014

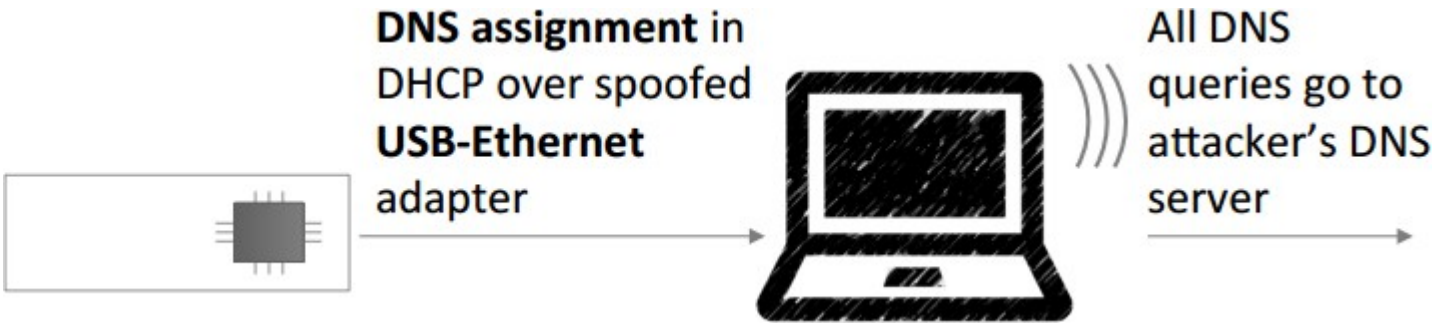
  BadUSB — On accessories that turn evil

  https://srlabs.de/badusb/



Keyboard emula[on is enough for infec[on and privilege escala[on (without need for software vulnerability)

Restart screensaver with password stealer added via an LD_PRELOAD library



- USB Redirection via RDP

  Easy Print / Drive Redirection / Smart Card Redirection

  Plug-and-Play Device Redirection / Input Redirection / Audio Redirection / Port Redirection

Source:USB attacks need physical access right? Andy Davis

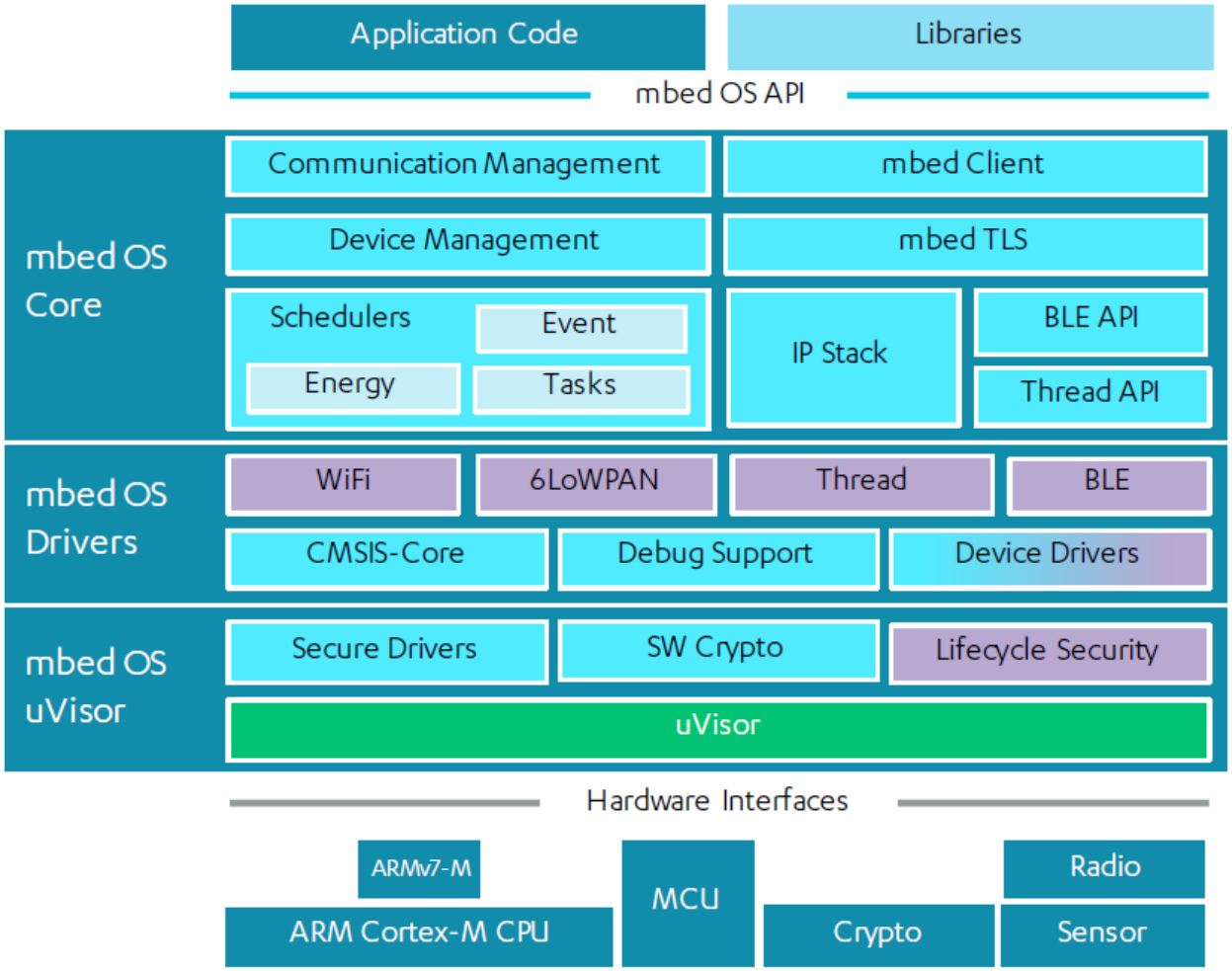
Users Really Do Plug in USB Drives They Find

- "end users will pick up and plug in USB flash drives they find by completing a controlled experiment in which we drop 297 flash drives on a large university campus."
- "We find that the attack is effective with an estimated success rate of **45–98%** and expeditious with the first drive connected in less than six minutes."
- Researchers at University of Illinois, Urbana Champaign, University of Michigan, Google, Inc.

Related talk at OpenIoT 2016
Handling Top Security Threats for Connected Embedded Devices - Eystein Stenberg, Mender

Source:
https://zakird.com/papers/usb.pdf

ARM mbed OS



Communication Security

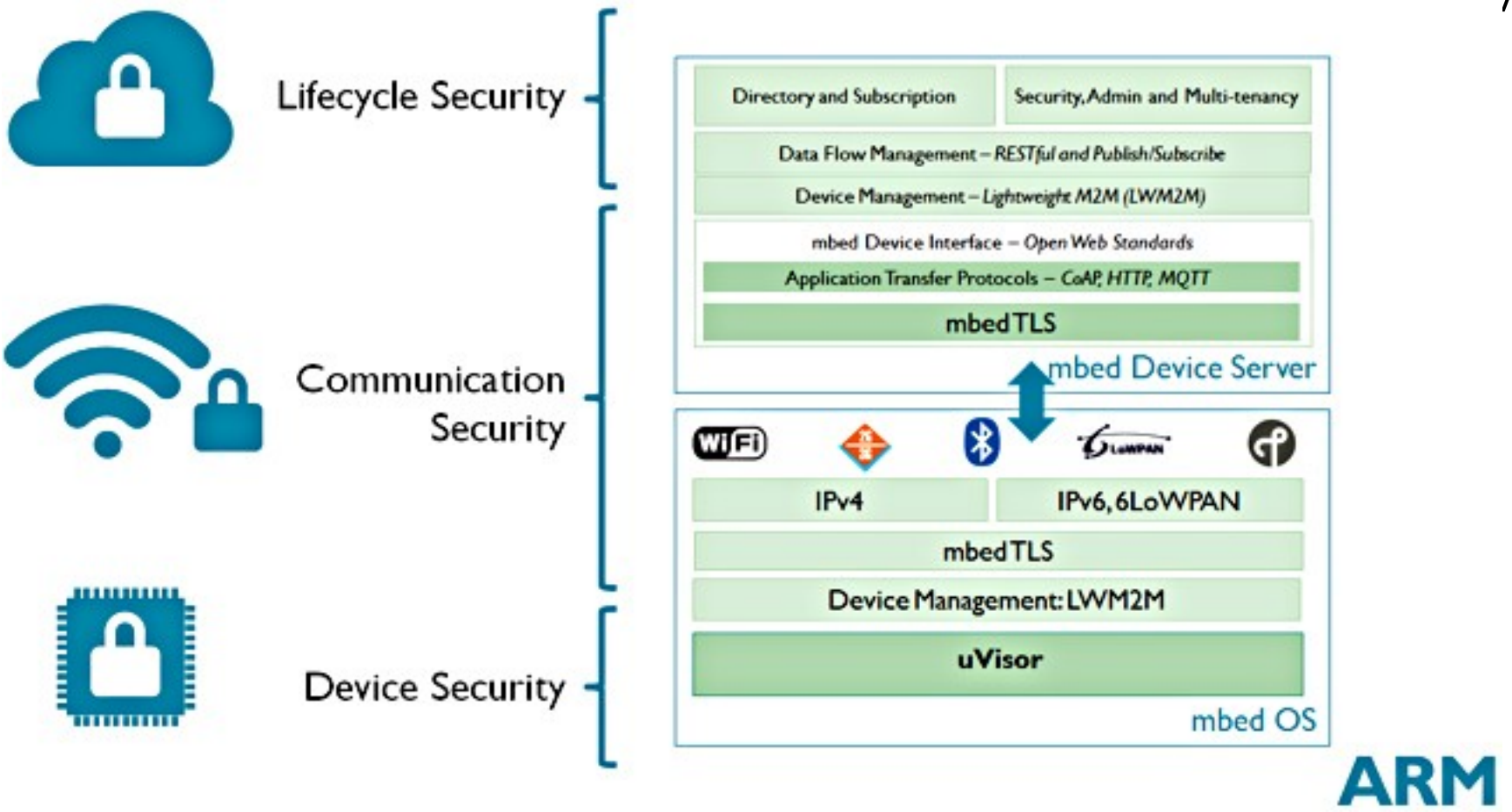
Lifecycle Security

Device Security

Source: source:http://www.slideshare.net/FoolsDelight/resilient-iot-security-the-end-of-flat-security-models

ARM

IoT Security



- ARM mbed OS
- ARM mbed uVisor
- ARM mbed TLS

- TrustZone in ARMv8-M
- security lifecycle management
- Apache License

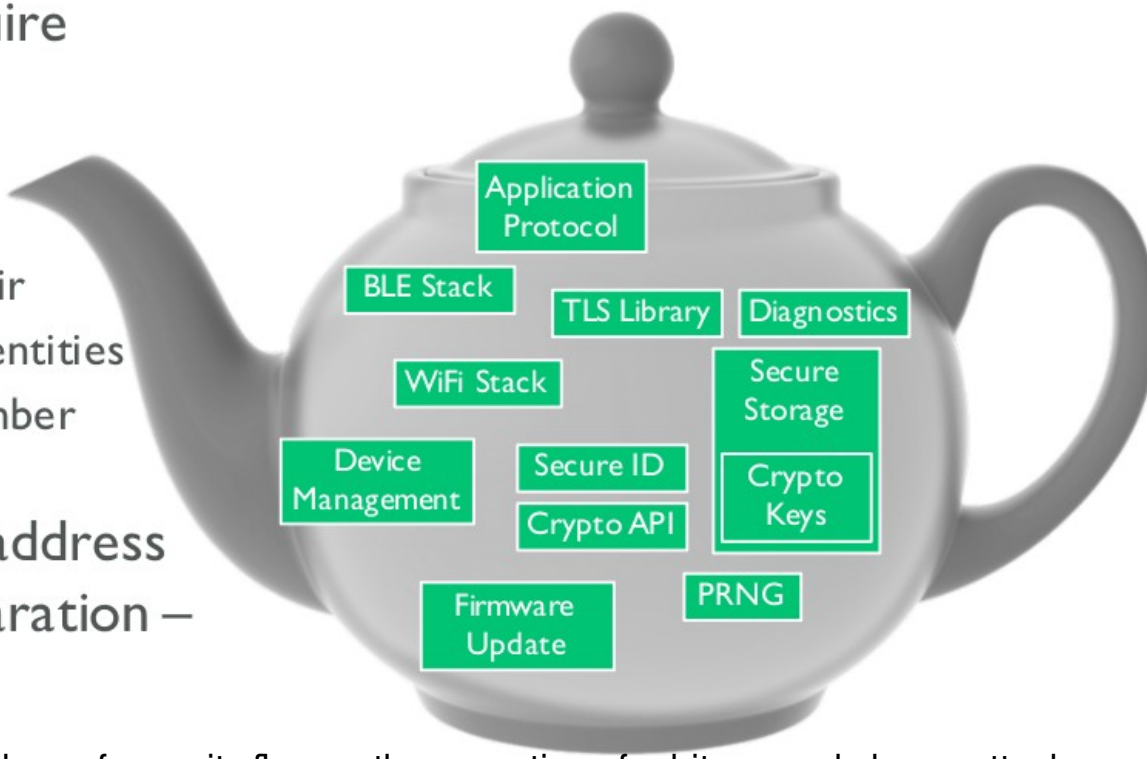
TrustZone for ARMv8-M

- Enables bus level protection in hardware
  - ARMv7-M requires software API filters for DMA access and other security critical operations
  - ARMv8-M can filter for DMA access for requests initiated by unprivileged code on bus level
- MPU banking reduces complexity of secure target OS
  - Secure OS partition own a private MPU with full control
  - OS keeps the privileged mode for fast IRQs
  - Fast interrupt routing and register clearing in hardware
  - Fast cross-box calls on TrustZone for ARMv8M – optimized call gateways

- because of the huge amount of code involved in maintaining WiFi connections or enabling ZigBee or BLE communication, the resulting attack surface is almost impossible to verify and therefore compromises device security

IoTeapot "Hello World" Example – The Attacker View

- Even simple IoT products require complex components
  - Secure server communication over complex protocols
  - Secure firmware updates over the air
  - Unclonable cryptographic device identities
  - Cryptography APIs and random number generation
- Existing IoT solutions use flat address spaces with little privilege separation – especially on microcontrollers

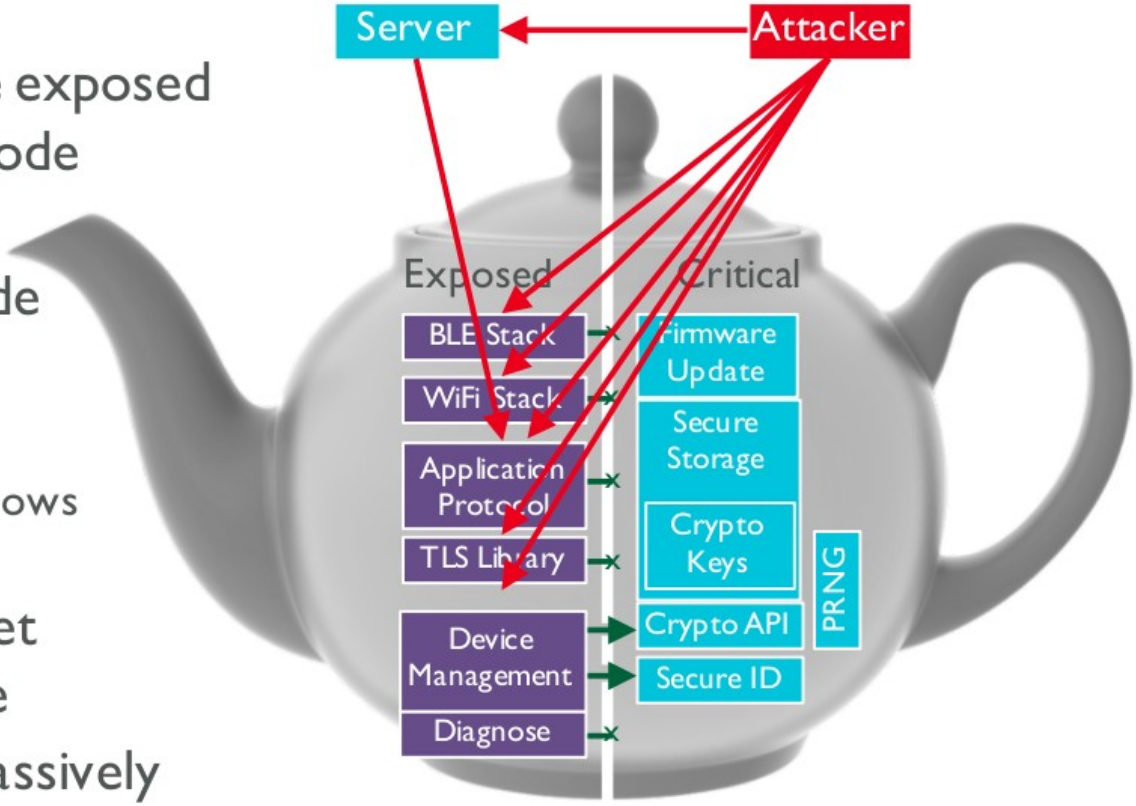


- the recovery from a common class of security flaws – the execution of arbitrary code by an attacker
  - Even a hardware-enforced root of trust and a secure boot loader will not fix that problem: the resident malware can run safely from RAM and block reset commands or flash erasing as part of a denial-of-service attack.

Source: ARM TechCon 2015

IoTeapot "Hello World" Example – Mitigation Strategies

- Attackers can compromise the exposed side without affecting critical code
- Using cryptographic hashes the integrity of the exposed side can be verified
  - Triggered on server request
  - Protected security watchdog box allows remote control
- Protected side can reliably reset exposed boxes to a clean state
- The device attack surface is massively reduced as a result



Source: ARM TechCon 2015

Security Impacts of mbed

- mbed OS allows user to develop applications over web
  - Developers may read or write memory over its address space mindlessly.
- IOT devices expose to Network/public
  - Attack through I/O
  - Cortex-M is Memory-mapped I/O
  - All configurations , including read and write through I/O are Memory issues.
  - Example: USART1_DR = 0x40011000 in STM32F429i

    All data go through USART1 need to access this address.

uVisor: Hardware-enforced security sandboxes

Security Functionality:
- Cryptography
- Key Management
- Secure FW Upgrade
- Secure Identity
- Security Monitoring

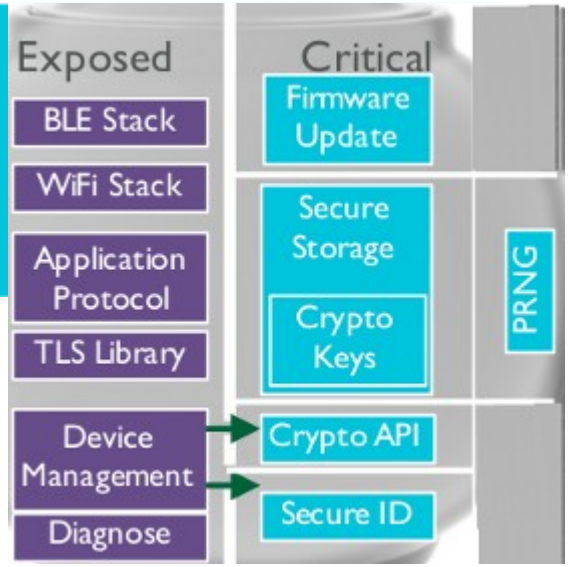
Isolated

Strong Separation

Remainder of mbed OS:
- HAL + Drivers
- Scheduler
- Connectivity Stack(s)
- Device Management
- User Application Code and Libraries

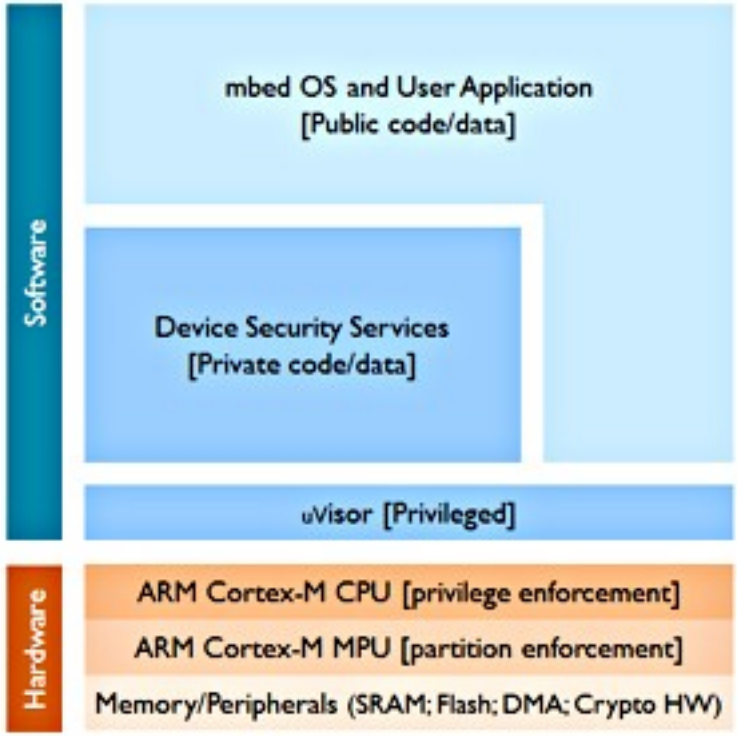
Non-critical

uVisor

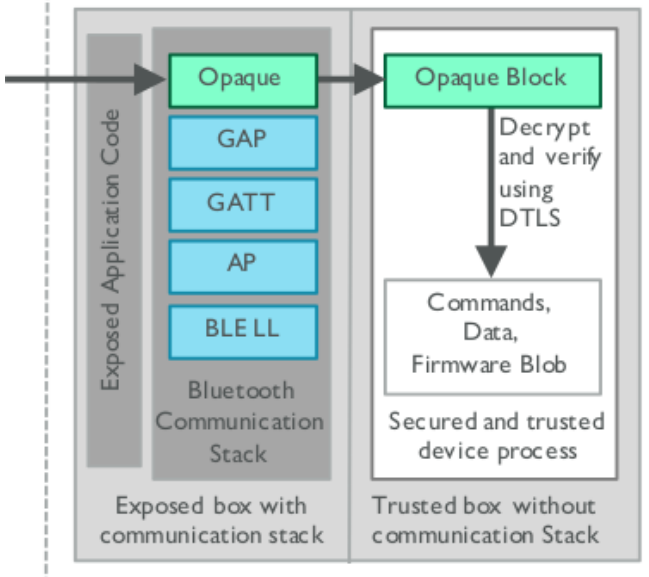


uVisor Design Principles

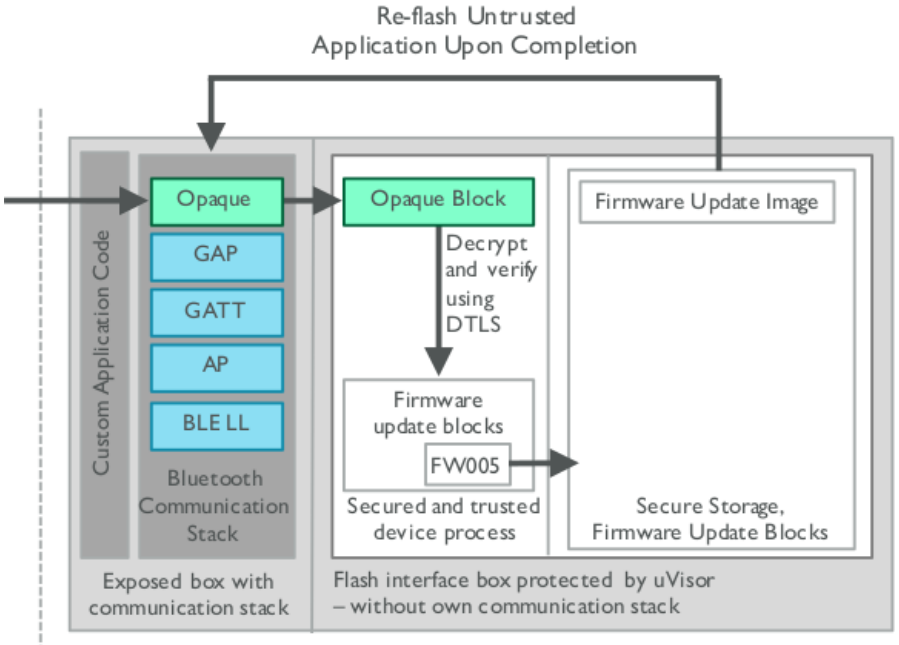
- Mutually distrustful security model
  - "Principle of Least Privilege"
  - Boxes are protected against each other
  - Boxes protected against malicious code from
  - broken system components, driver or other boxes
- Enforce API entry points across boxes
  - Box-APIs can be restricted to specific boxes
  - Allow interaction from the unpriviledged code by exposing system call-based APIs
- Per-box access control lists (ACL)
  - Restrict access to selected peripherals
  - Shared memories for box-box communication



- Box security not affected by communication stack exploits or infections outside of trusted box
- Resilient box communication over the available channels
  - Ethernet, CAN-Bus, USB, Serial
  - Bluetooth, Wi-Fi, ZigBee, 6LoWPAN



IoT Device owned by user.
Initial identity provisioned by System Integrator
Messages delivered agnostic of communication stack



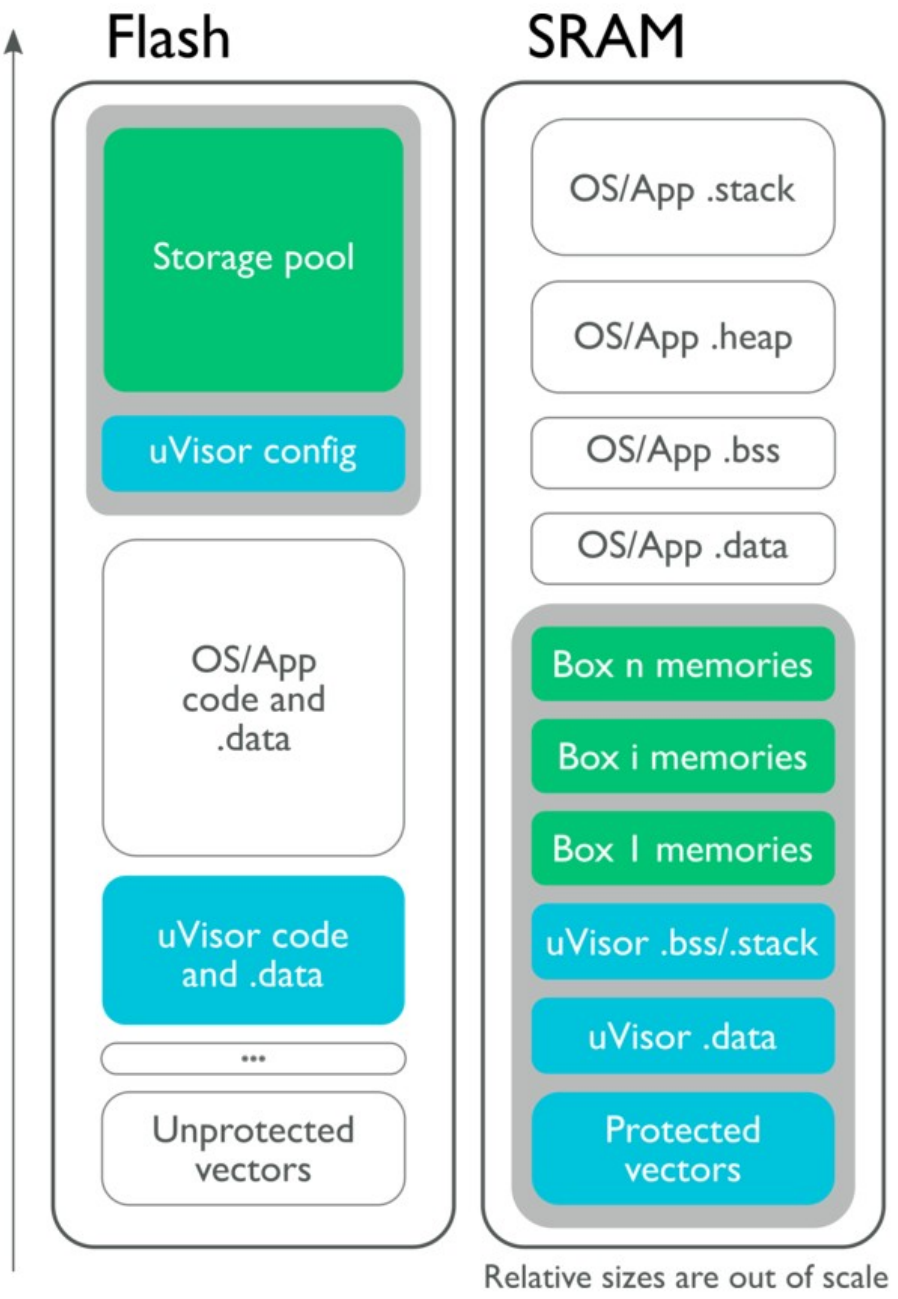
IoT device owned by user,
Initial identity provisioned by System Integrator,
Messages delivered independent of stacks

- Firmware manifest block augments existing firmware formats with safety and security features
- Crypto watchdog box enforces remote updates even for infected devices

Source: ARM TechCon 2015

MPU in ARMv7-M

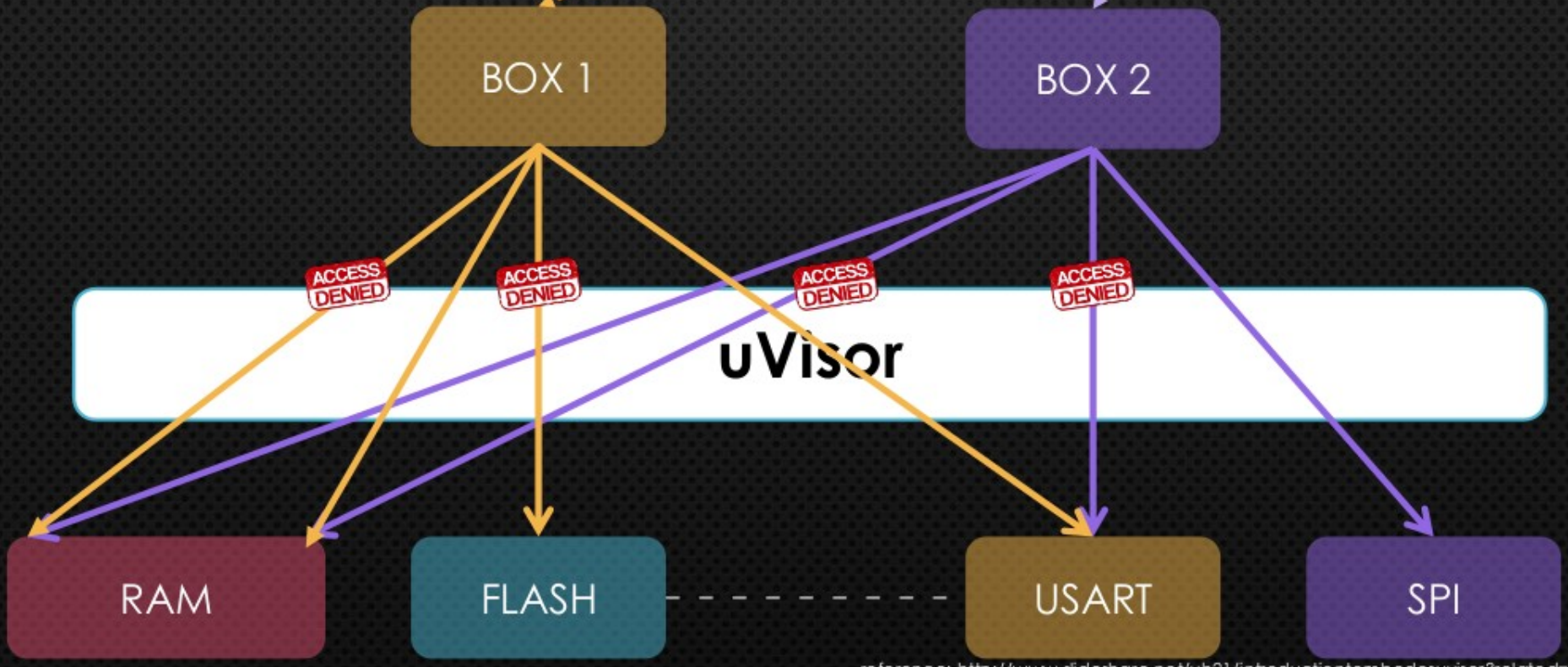
- Set Memory regions
  - Min: 32 bytes / Max: 4GB
- Set as XN
  - XN=Execute Never
  - cause MemManage Fault
- Read/Write
  - Privileged/Unprivileged
    - Read Only
    - Read/Write
    - No access
  - Denying access cause MemManage Fault
- Accessing MPU relative registers in unprivileged mode cause Bus Fault.



HOW TO PROTECT?

- Box n memories
- Box i memories
- Box 1 memories
- uVisor .bss/.stack
- uVisor .data
- Protected vectors

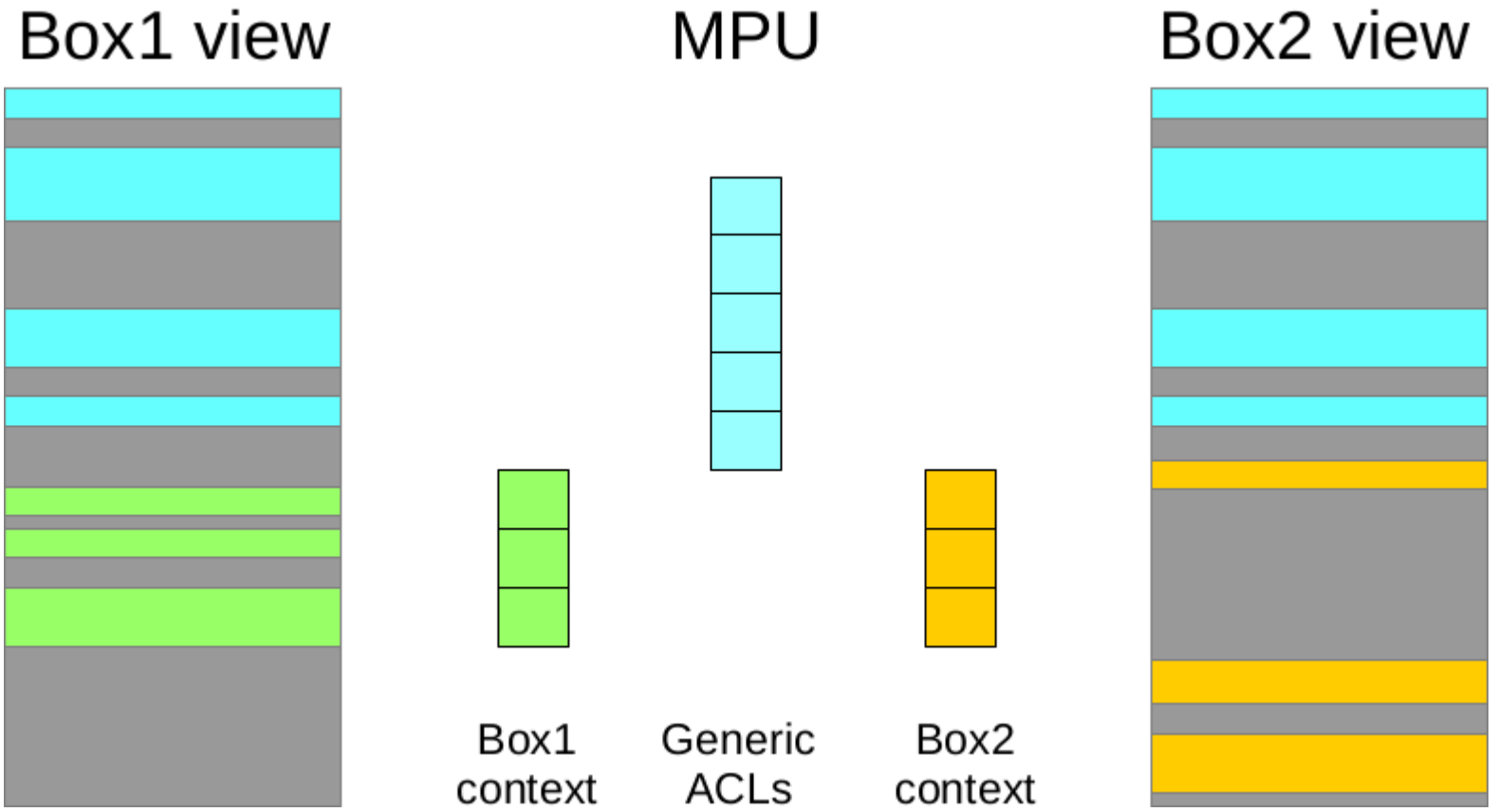
- Access Control Lists (ACLs)
  - Each color represents for one "user"
  - Each of them can only access its "belonings"
    - Otherwise, the MPU will cause it to get into "MemManage Fault"



reference: http://www.slideshare.net/vh21/introductiontombedosuvisor?related=1

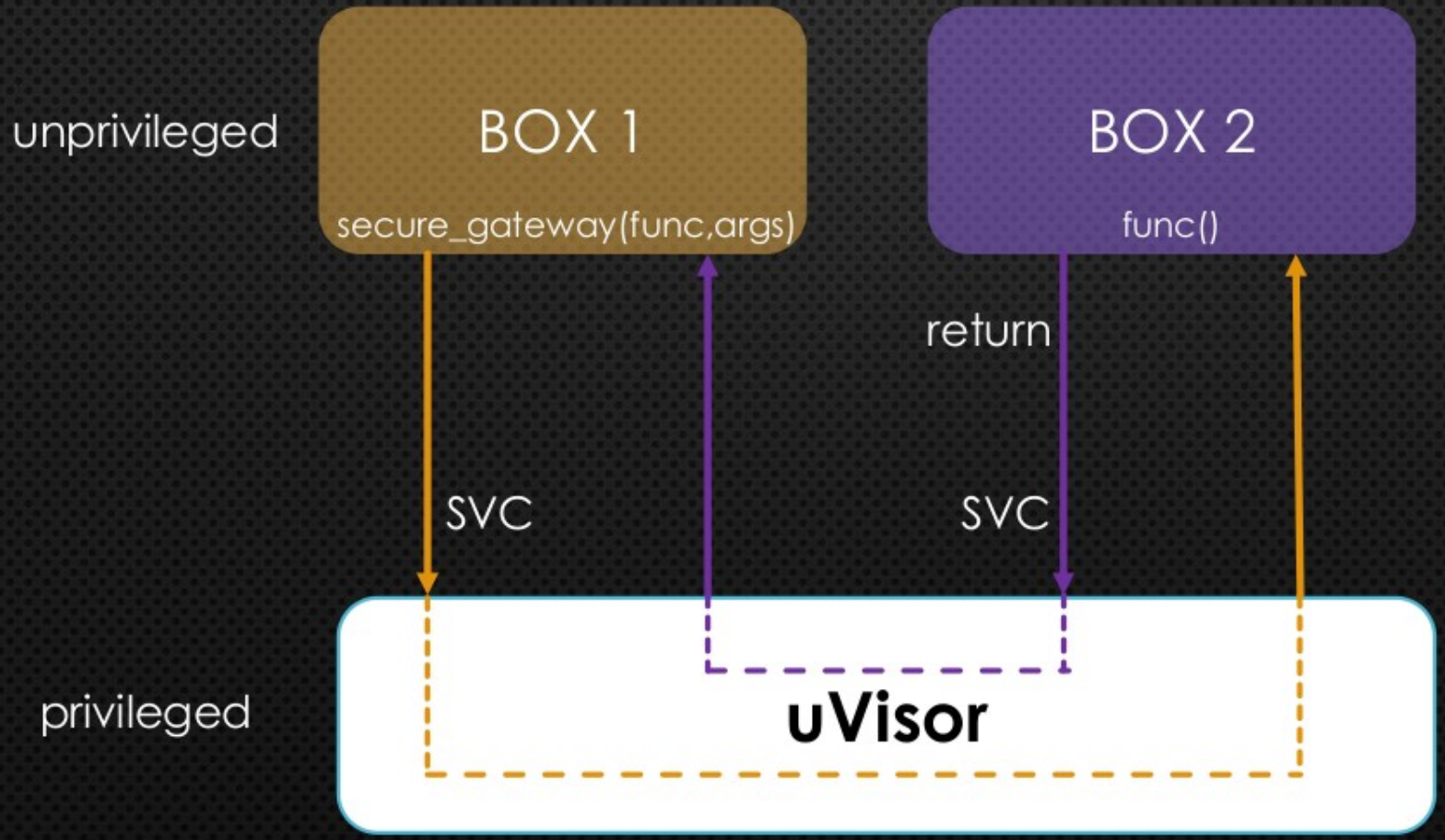
How ACLs are implemented

- ACLs and Box contexts isolation are implemented via MPU



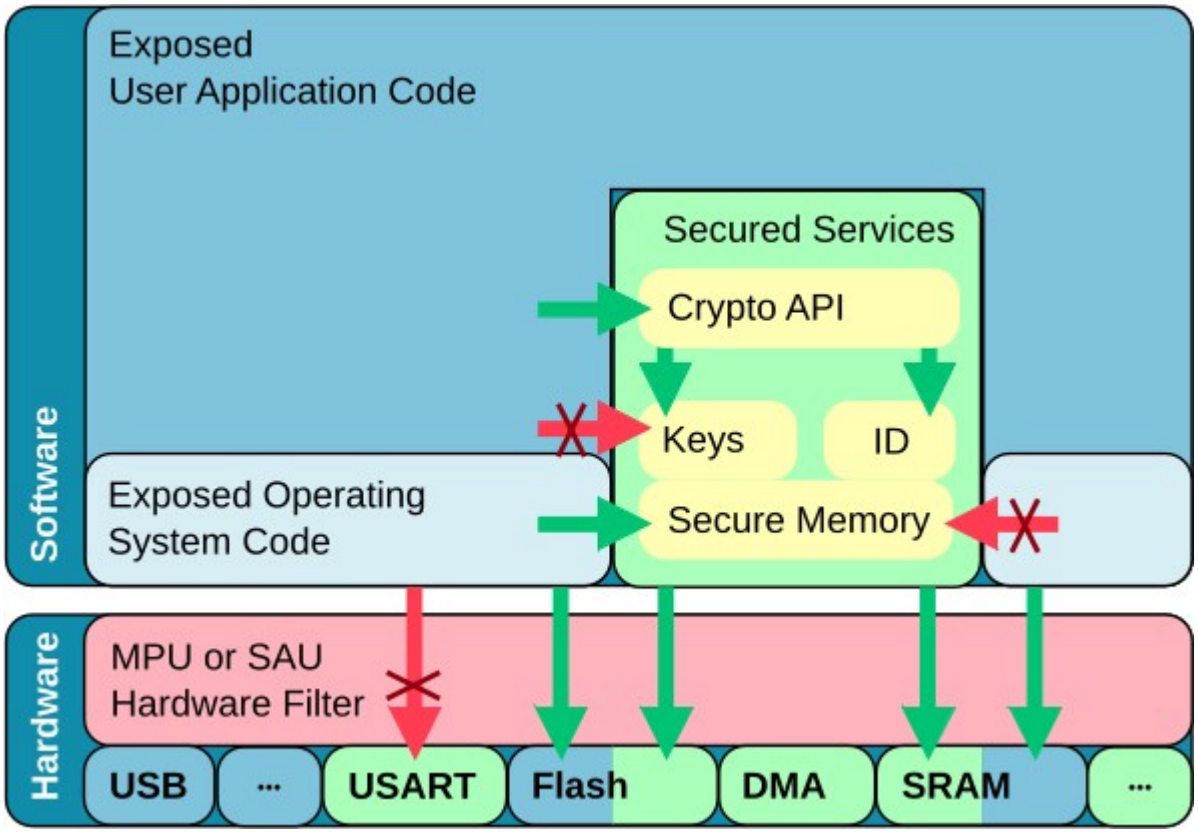
SECURE GATEWAY

for communication between boxes



reference: http://www.slideshare.net/vh21/introductiontombedosuvisor?related=1

uVisor Boot Sequence (ARMv7-M)



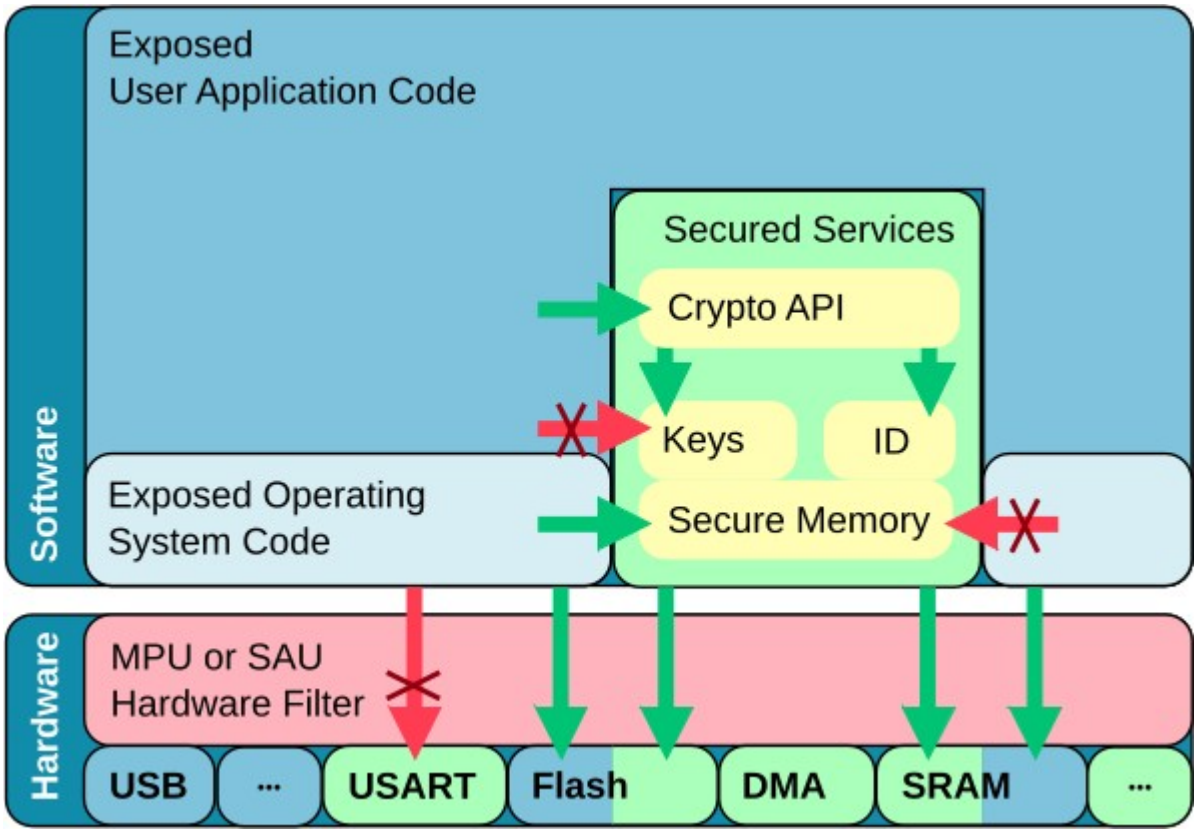
uVisor initialized first in boot process
→ Private stack and data sections
→ Private data sections in flash for storing secrets

Initialization of memory protection unit based on box ACL's
→ only necessary peripherals are accessible to box
→ Each box has private .bss data and stack sections

Relocation of interrupts vector table into secure memory

```
#include <uvisor-lib/uvisor-lib.h>
/* create background ACLs for the main box */
static const UvBoxAclItem g_background_acl[] = {
{UART0, sizeof(*UART0), UVISOR_TACL_PERIPHERAL},
{UART1, sizeof(*UART1), UVISOR_TACL_PERIPHERAL},
{PIT, sizeof(*PIT),UVISOR_TACL_PERIPHERAL}, };
UVISOR_SET_MODE_ACL(UVISOR_ENABLED, g_background_acl); /* set uvisor mode */
```

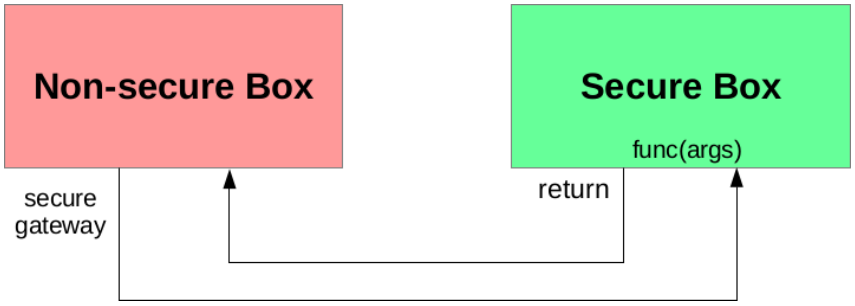
Protected Sandbox




```
/* private box context */
typedef struct {
uint8_t secret[SECRET_SIZE];
bool initialized;
} BoxContext;*/
```

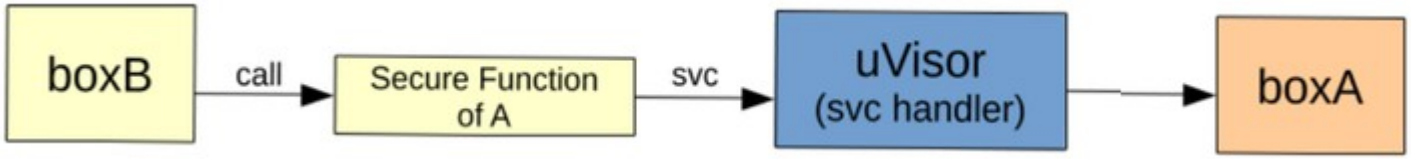
```
/* create ACLs for the module */
static const UvBoxAclItem g_box_acl[] = {
    {RNG, sizeof(*RNG), UVISOR_TACL_PERIPHERAL},
};

/* configure secure box compartment */
UVISOR_BOX_CONFIG(my_box_name, g_box_acl,
                  0x100 /* required stack size */, BoxContext);
```

Call Gateway



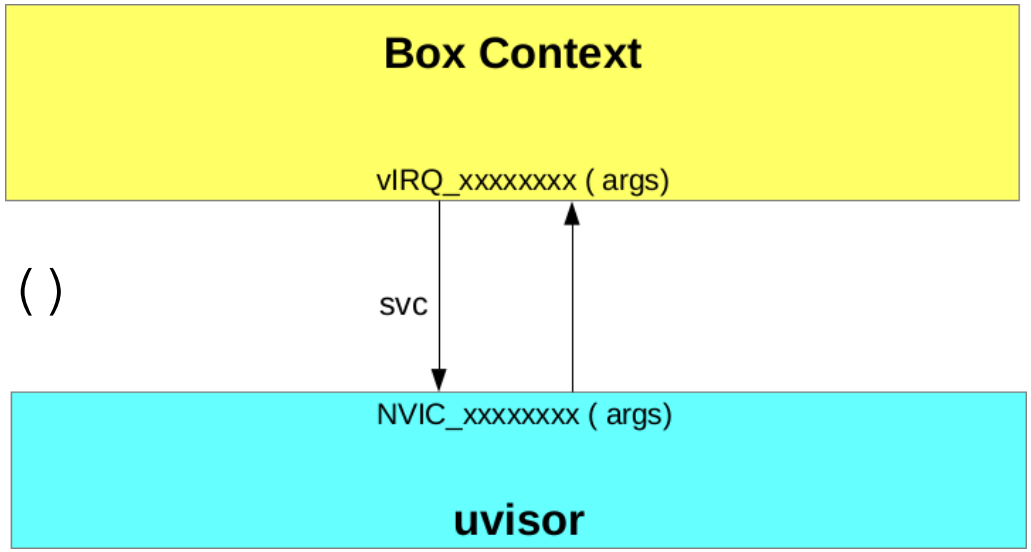
- Call gateways only accepted from flash memory
  - attacker has no write access to flash controller
- Metadata of call gateway at a fixed offset from uVisor gateway context switch – a supervisor call (SVC)
  - Contains pointer to target box configuration & target function
  - Guaranteed latency for cross-box calls
- Can limit access to specific caller boxes
- Security verified once during installation

```
/* the actual secure gateway */
#define secure_gateway(dst_box, dst_fn, ...)
({
    SELECT_ARGS(__VA_ARGS__)
    register uint32_t res asm("r0");
    asm volatile (
          "svc   UVISOR_API_SVC_CALL_ID\n"
          "b.n   skip_metadata%=\n"
          ".word UVISOR_SVC_GW_MAGIC\n"
          ".word dst_fn\n"
          ".word dst_box##_cfg_ptr\n"
          "skip_metadata%=:\n"
          : "=r" (res)
          : ASM_INLINE_ARGS(__VA_ARGS__)
    );
    res;
})
```



Interrupt Management APIs

- `vIRQ_SetVectorX()`
- `vIRQ_GetVector()`
- `vIRQ_EnableIRQ()`
- `vIRQ_DisableIRQ()`
- `vIRQ_ClearPendingIRQ()`
- `vIRQ_SetPendingIRQ()`
- `vIRQ_GetPendingIRQ()`
- `vIRQ_SetPriority()`
- `vIRQ_GetPriority()`
- `vIRQ_GetLevel()`



Interrupt Forwarding

Everything looks fine except debugging.

At present, uVisor supports quite limited approaches to debug

- LED patterns
  - used by default (!)
- Semihosting
  - based on SVC
  - I/O through GDB
- On-chip debugger
  - J-Link / ST-Link
  - wire connection

Error reason	RED LED blinks
PERMISSION_DENIED	1
SANITY_CHECK_FAILED	2
NOT_IMPLEMENTED	3
NOT_ALLOWED	4
FAULT_MEMMANAGE	5
FAULT_BUS	6
FAULT_USAGE	7
FAULT_HARD	8
FAULT_DEBUG	9

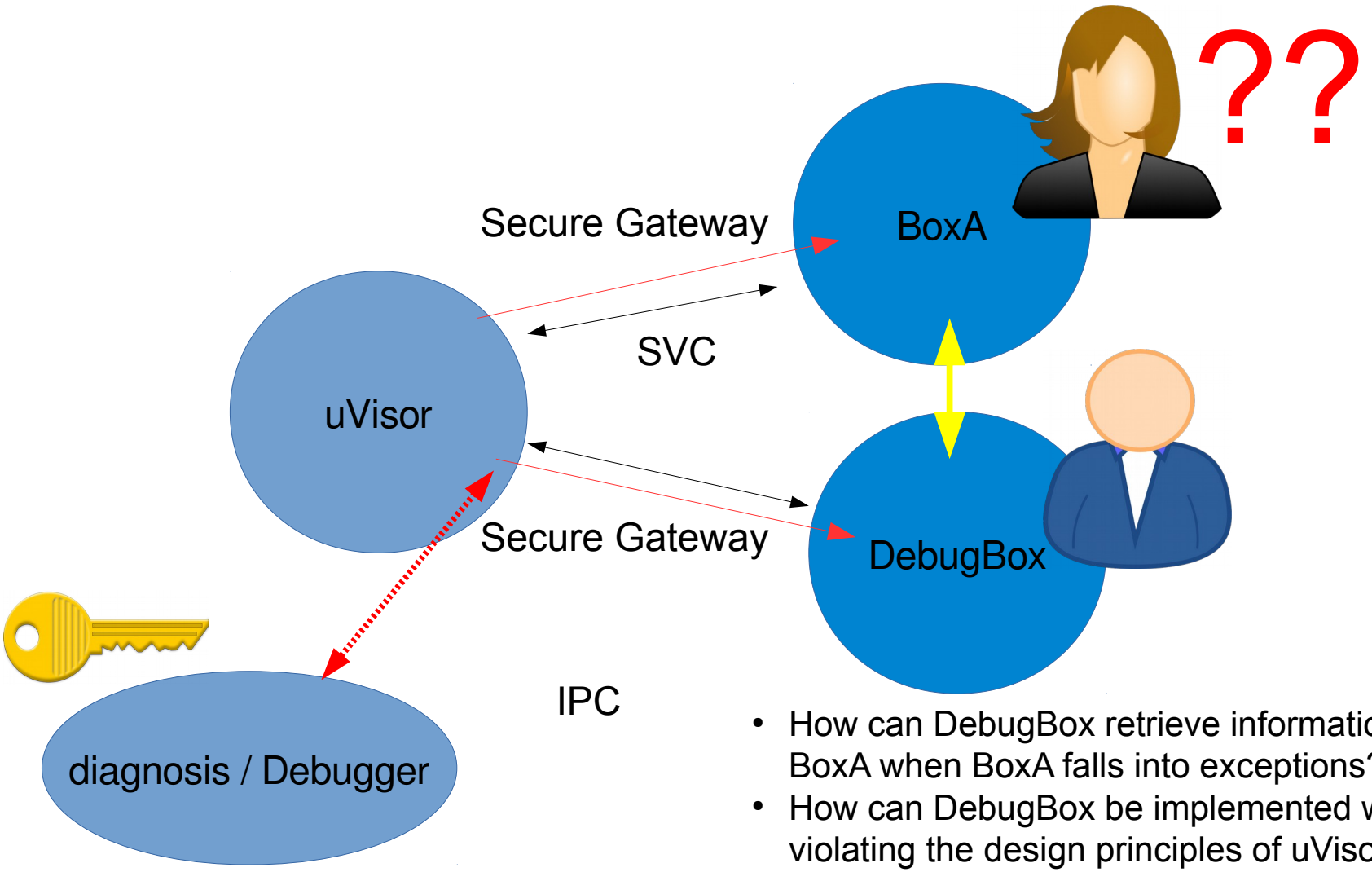
GDB usage through wireless communications!

- Look up
  - Memory
  - registers
- Control execution
  - Singel Step
  - Single Instruction
  - Breakpoint
  - Watchpoint

at any time and any where.

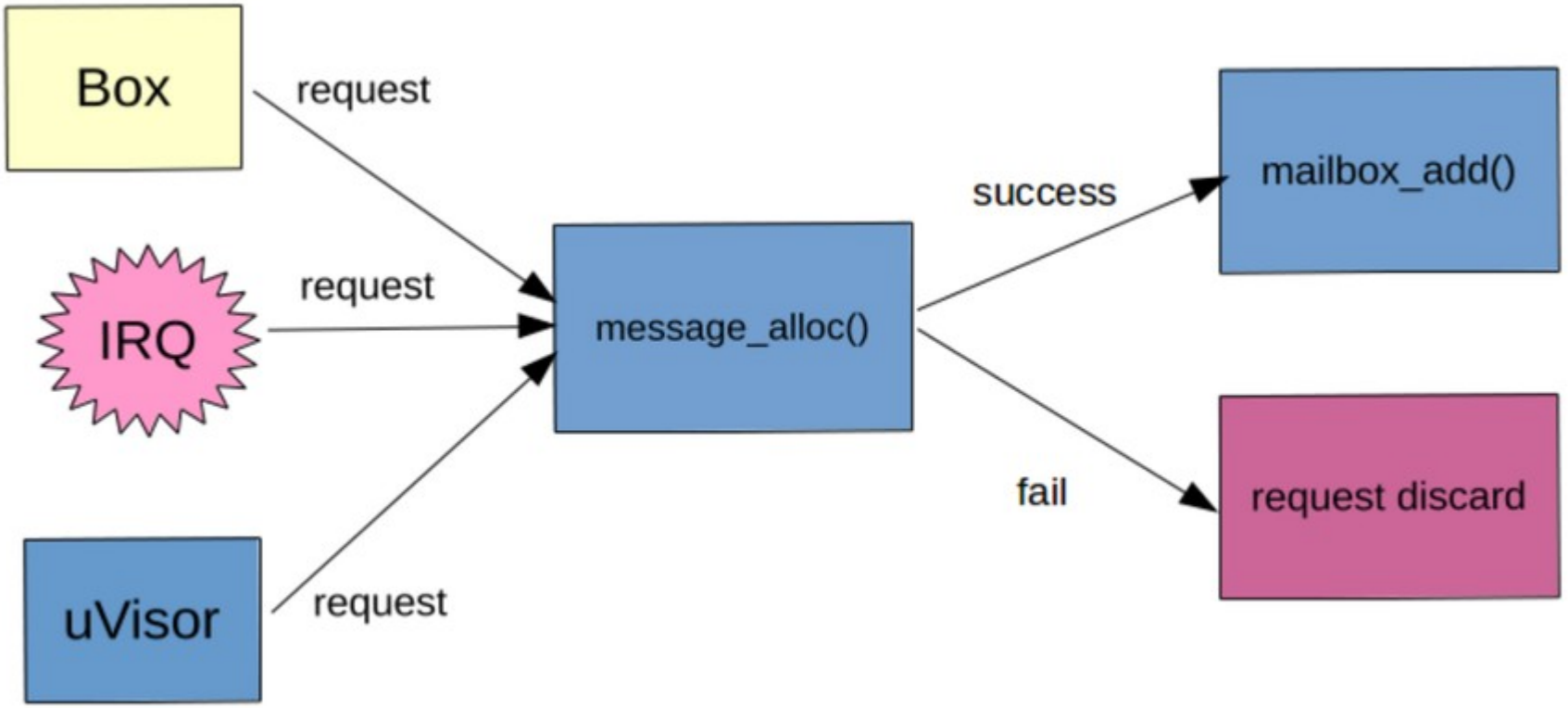
```
(gdb) b main.cpp:44
Breakpoint 1 at 0x8000a5e: file main.cpp, line 44.
(gdb) where
#0  us_ticker_read () at ../../external/mbed/libraries/mbed/targets/hal/TARGET_STM/TARGET_STM32F4/us_ticker.c:50
#1  0x0800379e in wait_us (us=500000) at ../../external/mbed/libraries/mbed/common/wait_api.c:29
#2  0x08003766 in wait (s=0.5) at ../../external/mbed/libraries/mbed/common/wait_api.c:20
#3  0x08000a5e in main () at main.cpp:43
(gdb) c
Continuing.

Breakpoint 1, main () at main.cpp:44
44                      myled = 0;
(gdb) p/x i
$1 = 0x1
```

DebugBox



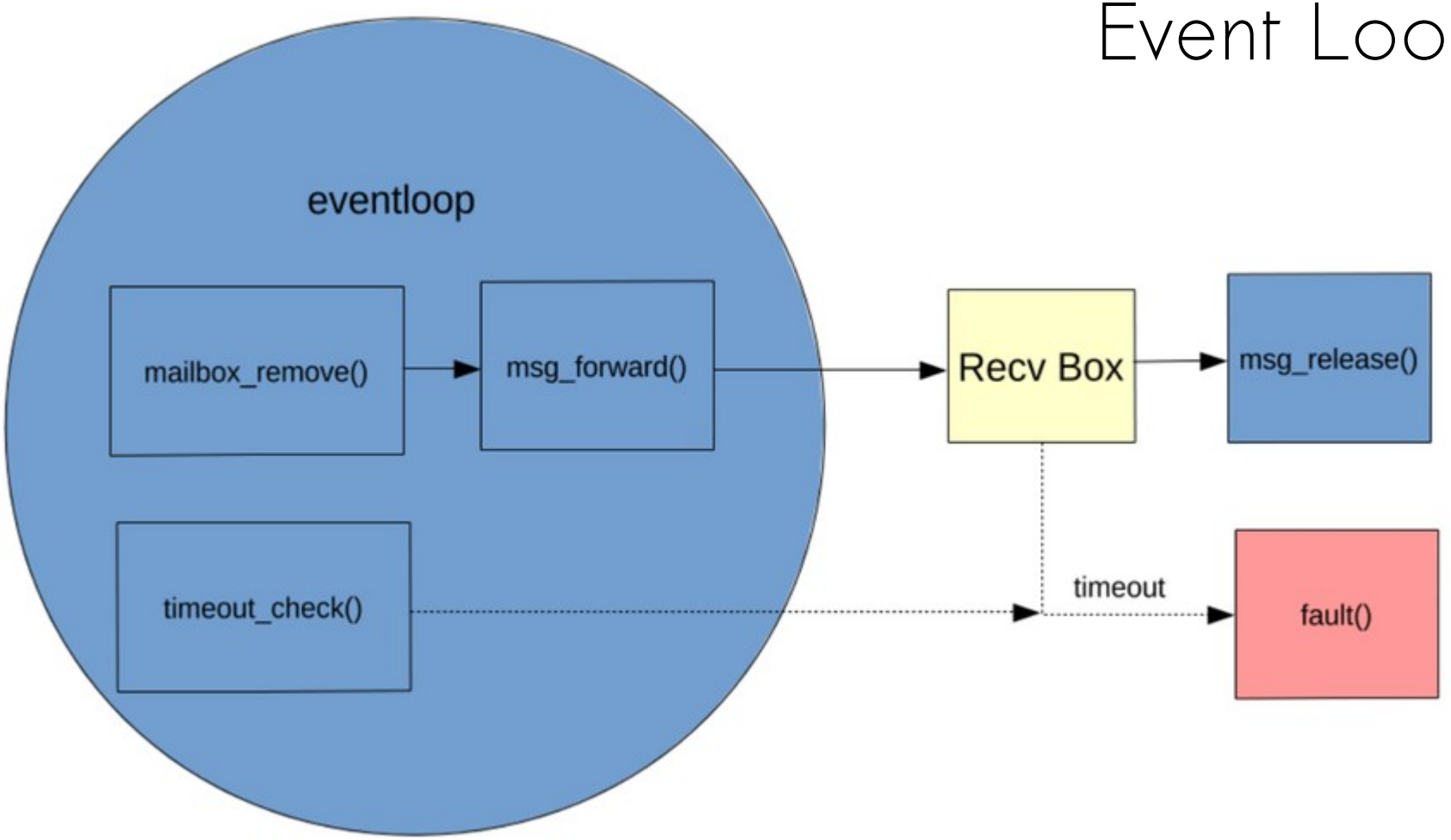
- How can DebugBox retrieve information from BoxA when BoxA falls into exceptions?
- How can DebugBox be implemented without violating the design principles of uVisor?

Mailbox communications



- The communication request instead of being transmitted immediately and delivered at an appointed time/at a desirable time, it is buffered in mailbox. The mailbox can be divided into two components: message queue and eventloop.

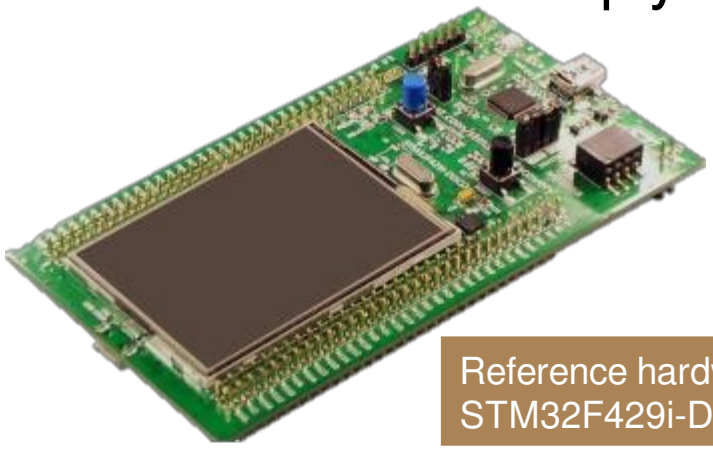
Event Loop



- Once Recv box completes its handling, the resource must return immediately and the initial box needs restoration. Furthermore, to avoid resource holding permanently, there is also a mechanism to recycle the resource automatically.

Integration of external projects

- CrashDebug
  - enable post-mortem debugging of Cortex-M crashes with GDB.
- CrashCatcher
  - catch Hard Faults on Cortex-M devices and save out a crash dump; used by CrashDebug
- MRI
  - GDB compatible debug monitor for Cortex-M devices.
  - Running over any of the UART ports on the device.
  - Do not reply on On-Chip debugger hardware



Reference hardware:
STM32F429i-Discovery

dashed line represents for any communication way,such as USART or Bluetooth.

Reference hardware:
STM32F429i-Discovery

Ad-Hoc Debugging

future framework between debugger and debuggee

System 1

remote GDB

GDB
remote serial protocol

Application BOX(s)
with
access permission
in the ACLs of the
Debug Box

Debug Box

MRI

CrashCatcher

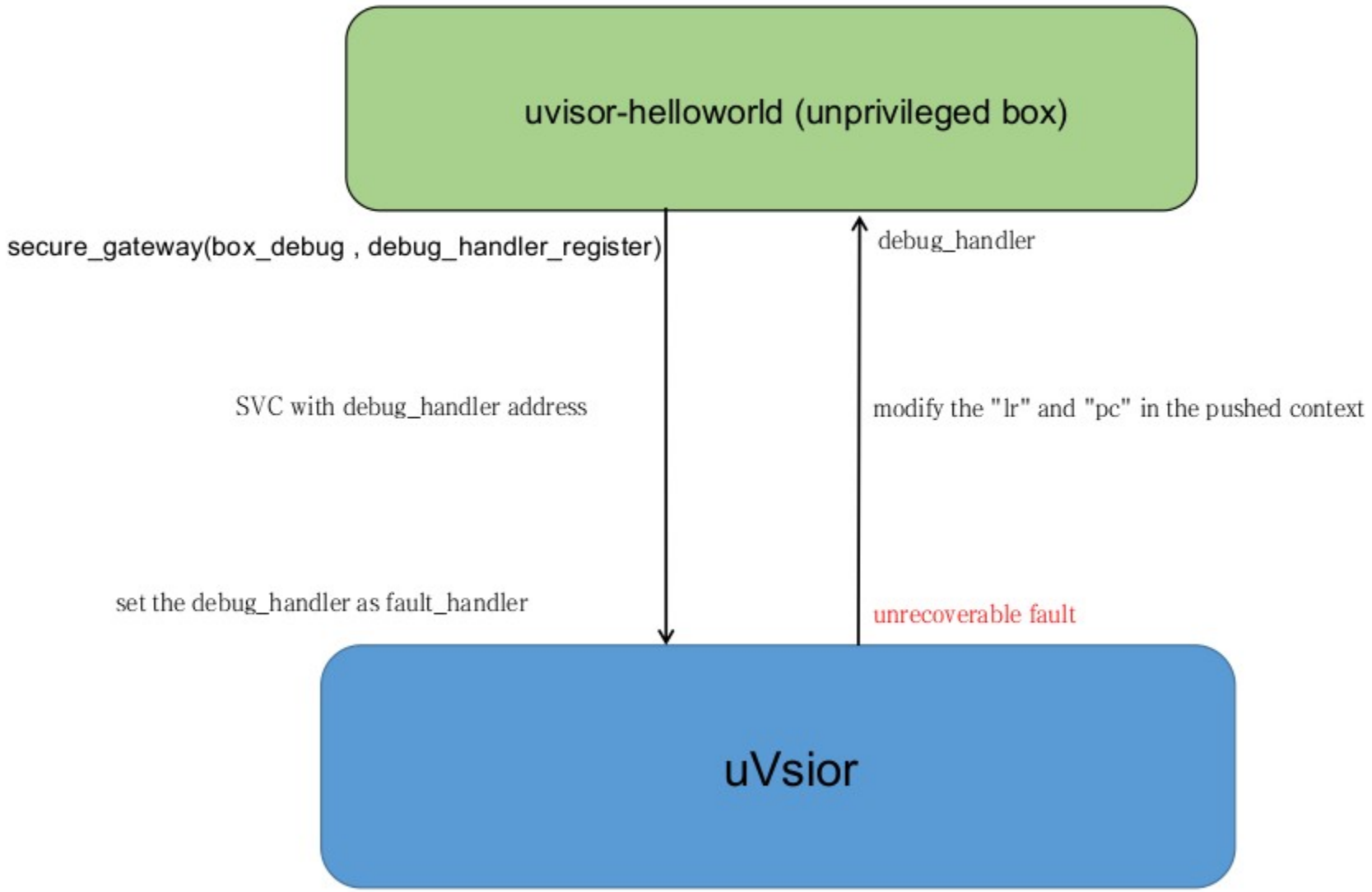
System 2

Save
CrashCatcher
dump

GDB with
CrashDebug

uVisor

DebugBox Flow



CrashCatcher

- Save the memory content in the hardfault_handler
  - used by GDB + CrashDebug
  - send the content to remote host or save in the local flash memory.
- The format must be readable by GDB with CrashDebug
  - Little-Edian
  - registers content
  - StartAddress - EndAddress
  - Content

```
63430200
00000000
74020020000000000ED00E000000000
00000000000000000000000000000000
00000000000000000000000000000000
02000000
D0FF0220
950A0008A80B000800000021
03000020
0000002000C00120
00000320A15D0008ED5D0008FD0C0008
2B1F00082D1F00082F1F000800000000
00000000000000000000000ED5D0008
331F000800000000ED5D0008ED5D0008
ED5D0008ED5D0008ED5D0008ED5D0008
ED5D0008ED5D0008ED5D0008ED5D0008
ED5D0008ED5D0008ED5D0008ED5D0008
...
```

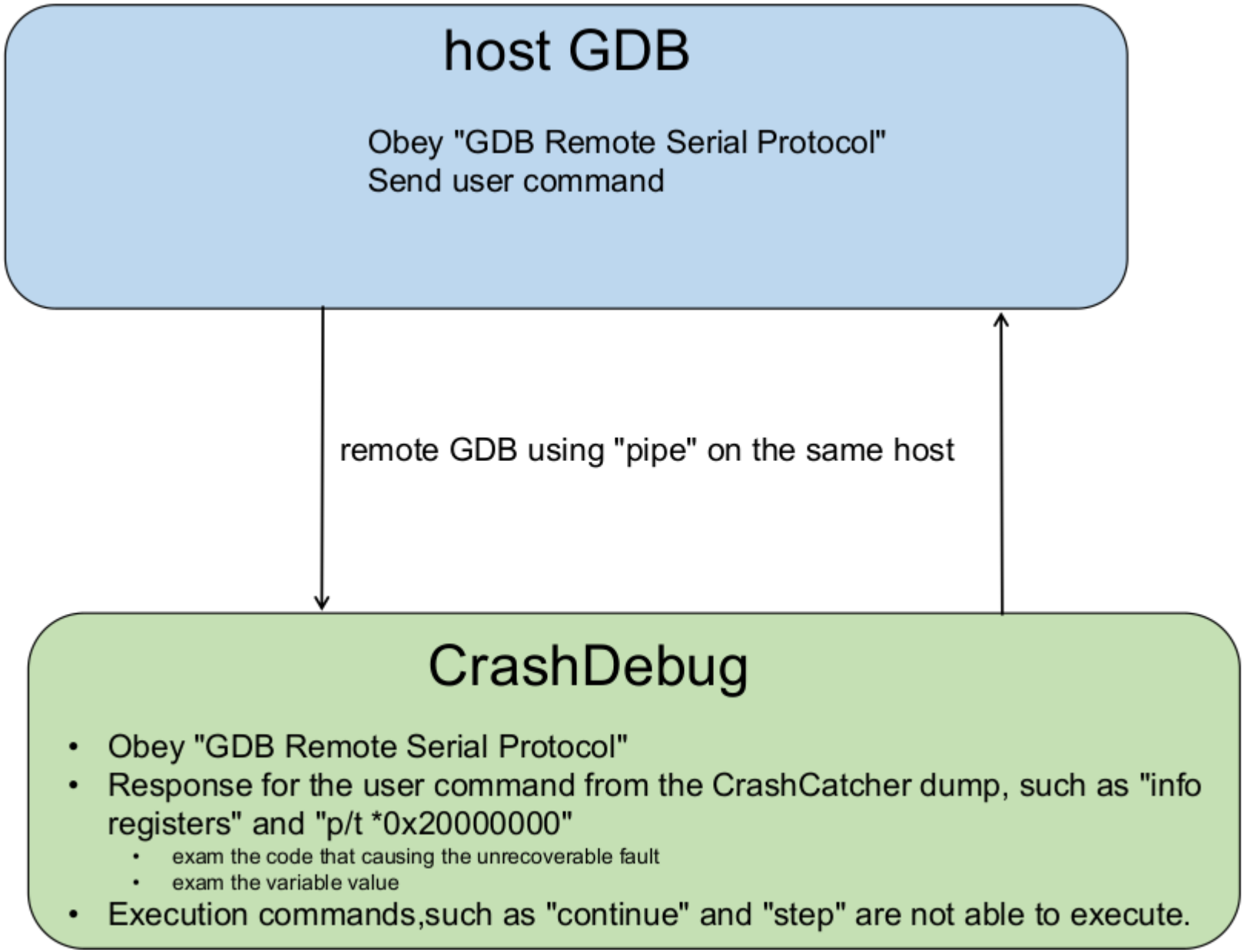
CrashDebug

- Post-mortem debug

  With the crashed dump memory content, we can
  - let the GDB view it as an alive target.
  - use GDB commands to examine the critical variable value.
  - locate where the problem results from
- Full backtrace

```
63430200
00000000
7402002000000000ED00E000000000
00000000000000000000000000000000
00000000000000000000000000000000
02000000
D0FF0220
950A0008A80B000800000021
03000020
0000002000C00120
00000320A15D0008ED5D0008FD0C0008
2B1F00082D1F00082F1F000800000000
00000000000000000000000ED5D0008
331F000800000000ED5D0008ED5D0008
ED5D0008ED5D0008ED5D0008ED5D0008
ED5D0008ED5D0008ED5D0008ED5D0008
ED5D0008ED5D0008ED5D0008ED5D0008
...
```

CrashDebug


host GDB
Obey "GDB Remote Serial Protocol"
Send user command
remote GDB using "pipe" on the same host
CrashDebug
• Obey "GDB Remote Serial Protocol"
• Response for the user command from the CrashCatcher dump, such as "info registers" and "p/t *0x20000000"
• exam the code that causing the unrecoverable fault
• exam the variable value
• Execution commands,such as "continue" and "step" are not able to execute.

MRI (Monitor for Remote Inspection)

- Allow to use GDB remote debugging through ANY communication methos
  - without the need of On-Chip debugger
  - Currently support USART in STM32F429i-Discovery Cortex-M4 devices.
- GDB Remote Serial Protocol
  - communicate with host GDB.
    - retrieve commands by modifying USART handler
  - According to the commands sent from host GDB, MRI sets the debug monitor in Cortex-M devices.
- Debug Monitor
  - Halt mode
  - debug monitor: based on exception handler

Conclusion

- The device-side security solution in mbed OS is uVisor, implemented as a reasonable software segmentation strategy purely in software without requiring additional hardware features.
- uVisor implements sandbox via MPU, which allows sectioning off regions of memory and defining the protection at the task level.
- Debugging IoT devices through wireless communications is different traditional software development model. We have to introduce smart ways for comprehensive debugging and profiling.

Reference

- Resilient IoT Security: The end of flat security models, Milosch Meriac, ARM TechCon 2015
- smart solutions for the internet of things, Genesi USA, Inc.
- Introduction to mbed-OS uvisor, Viller Hsiao
- ARMlock: Hardware-based Fault Isolation for ARM, North Carolina State University / Xi'an Jiaotong University / Florida State University
- Mactans: Injecting Malware into iOS Devices via Malicious Chargers
- CrashDebug & MRI, Adam Green: https://github.com/adamgreen