Resilient IoT Security: The end of flat security models

ARM

Milosch Meriac
IoT Security Engineer
milosch.meriac@arm.com

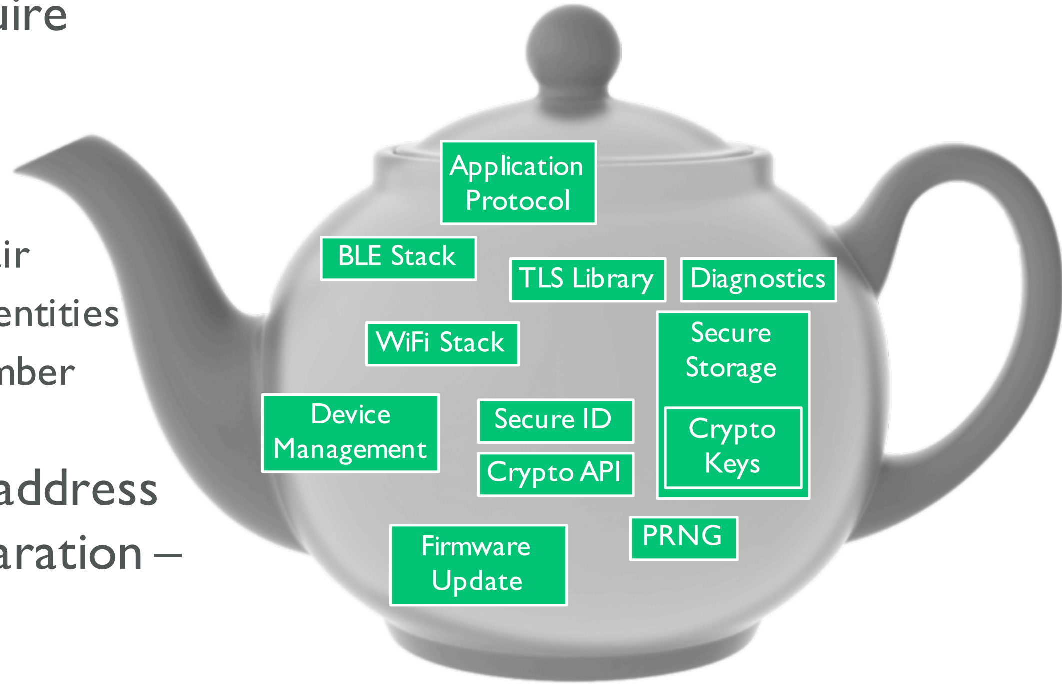
"Securing a computer system has traditionally been a battle of wits: the penetrator tries to find the holes, and the designer tries to close them."

– Morrie Gasser, Author of "Building a Secure Computer System"

© ARM 2015

IoTeapot "Hello World" Example – The Attacker View

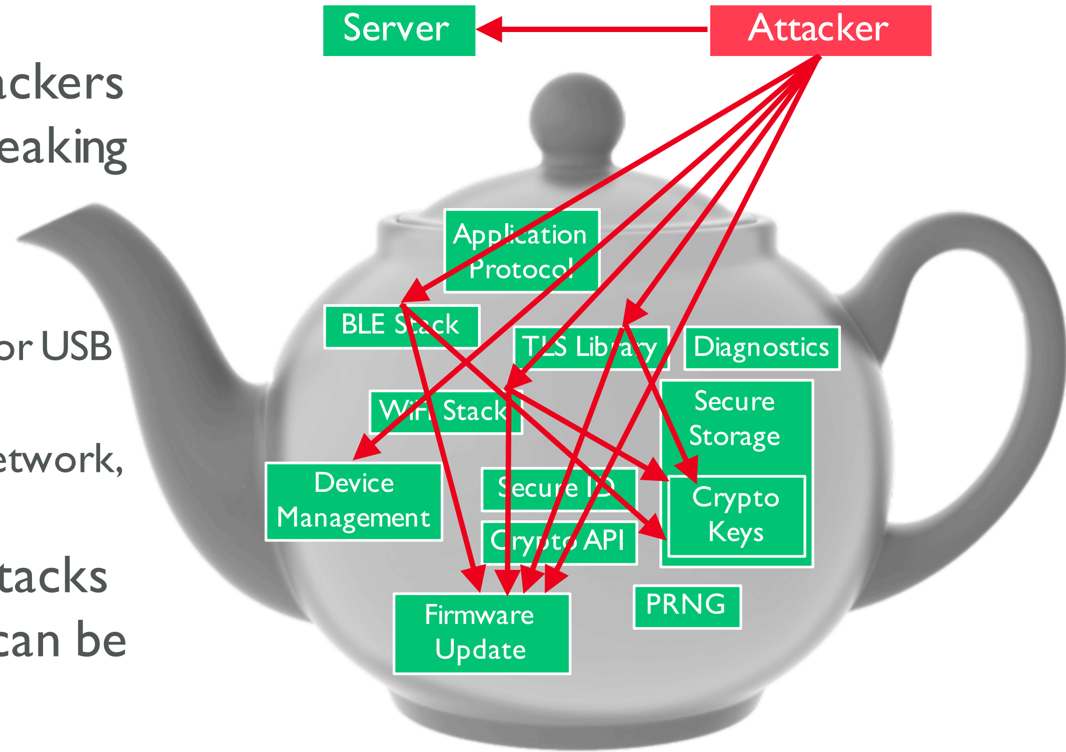
- Even simple IoT products require complex components
  - Secure server communication over complex protocols
  - Secure firmware updates over the air
  - Unclonable cryptographic device identities
  - Cryptography APIs and random number generation
- Existing IoT solutions use flat address spaces with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



© ARM 2015
ARM

IoTeapot "Hello World" Example – The Attacker View

- Flat security models allow attackers to break device security by breaking any system component
- Common attack entry points:
  - Complex protocols like TLS, Wi-Fi or USB device configuration
  - Firmware update functions (USB, network, CAN…)
- Impossible to recover from attacks as firmware update functions can be compromised by the attacker



© ARM 2015
ARM

“It ain’t what you don’t know that gets you into trouble. It’s what you know for sure that just ain’t.”

– Mark Twain

© ARM 2015

Security Plus Time Equals Comedy: Plan for the Worst Case

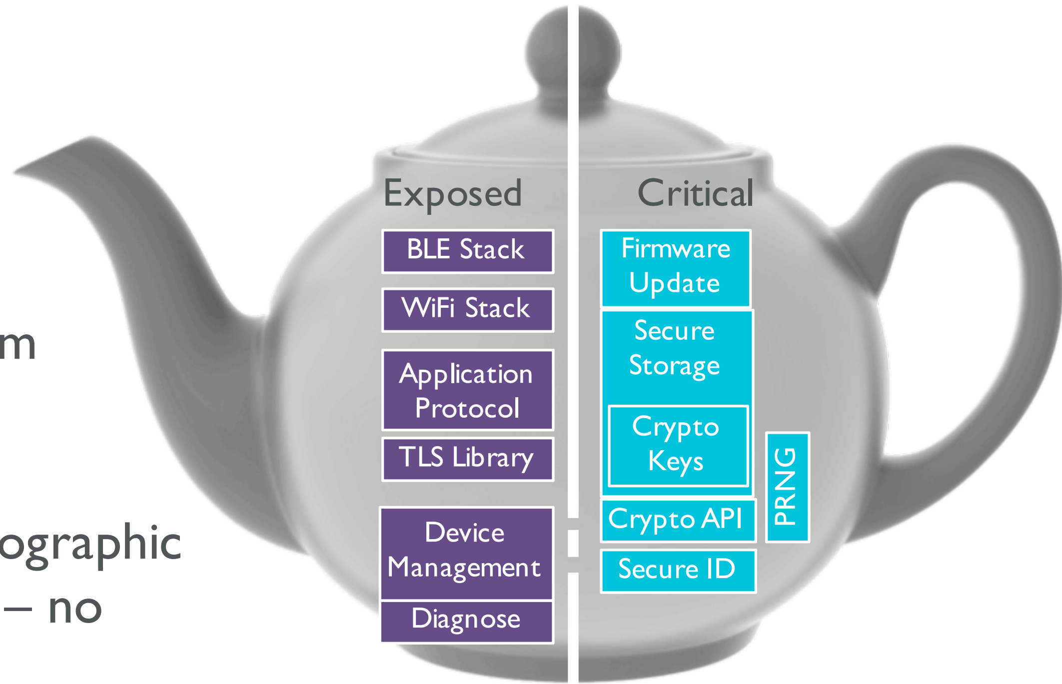
- System security is dynamic over device lifetime
- Devices last longer than expected
- Likelihood of attacks underestimated as a result
- If your system is successful, it will be hacked
- Deployment costs of firmware updates in case of hacks often surpasses costs of new devices. As a result known-broken systems are kept in use
- Developers must ensure secure, reliable and "cheap" update possibilities
- Devices must be capable of remote recovery from an untrusted state back to a trusted state

https://commons.wikimedia.org/wiki/File:Cret_Comedy_and_Tragedy.JPG

© ARM 2015
ARM

IoTeapot "Hello World" Example – Mitigation Strategies

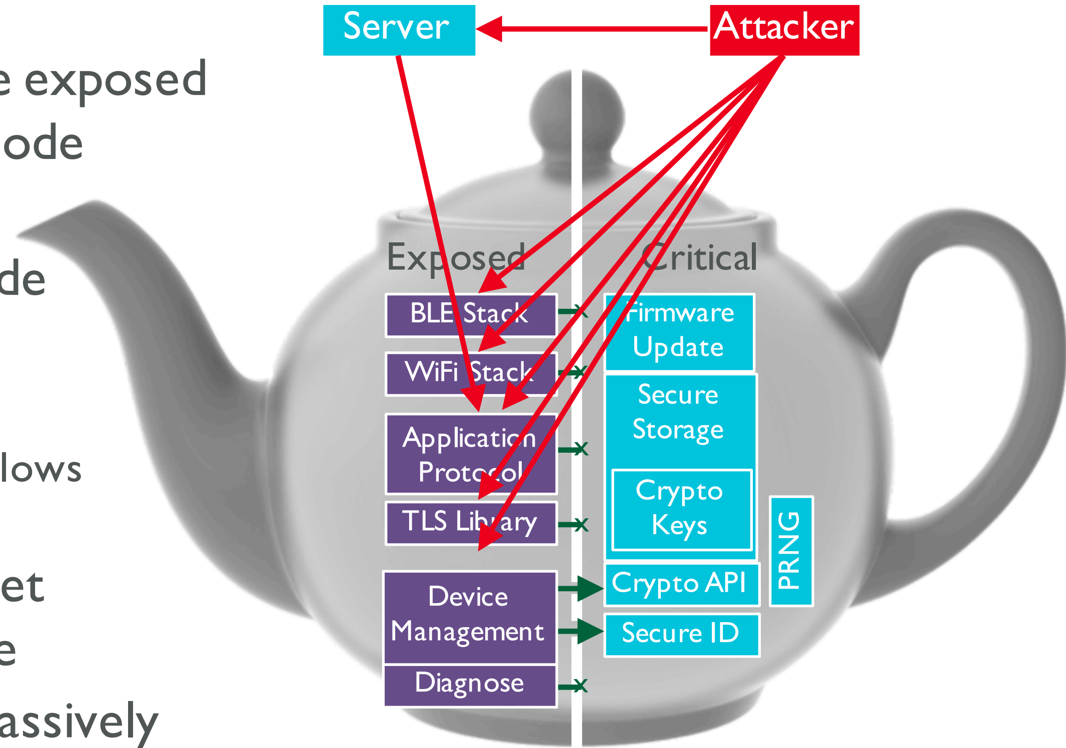
- Split security domains into
  - exposed uncritical code
  - protected critical code
- Keep footprint of critical code small to enable verification
- Protect key material and system integrity using the ARMv7-M hardware memory protection
- Public code operates on cryptographic secrets via defined private API – no access to raw keys



© ARM 2015
ARM

IoTeapot "Hello World" Example – Mitigation Strategies

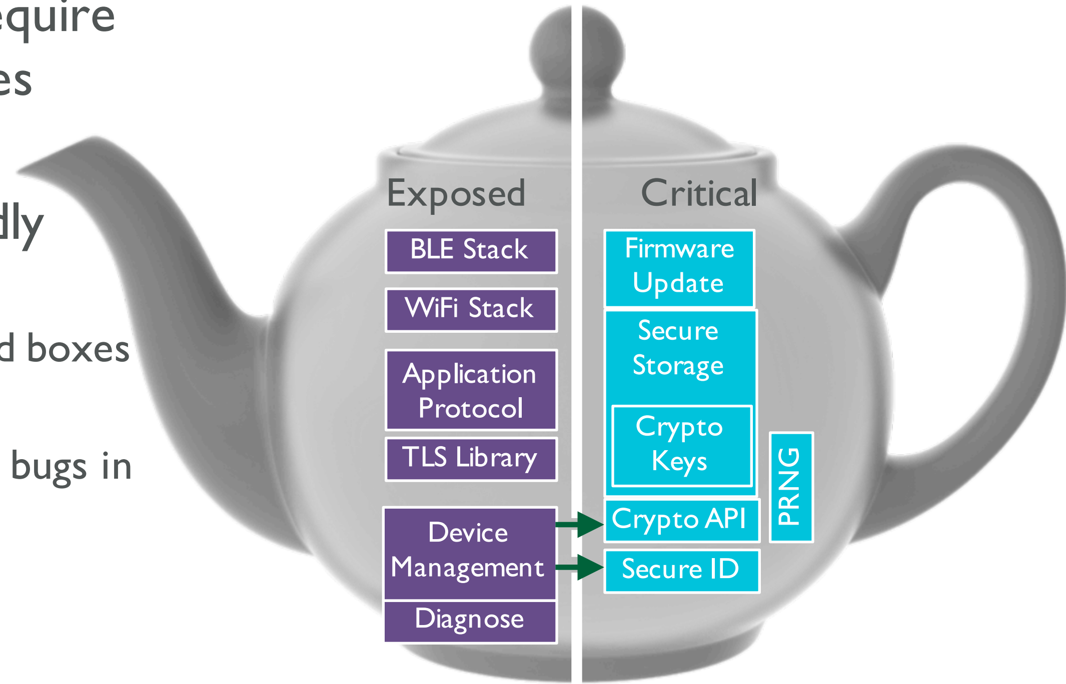
- Attackers can compromise the exposed side without affecting critical code
- Using cryptographic hashes the integrity of the exposed side can be verified
  - Triggered on server request
  - Protected security watchdog box allows remote control
- Protected side can reliably reset exposed boxes to a clean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



© ARM 2015
ARM

Enable Fast Innovation

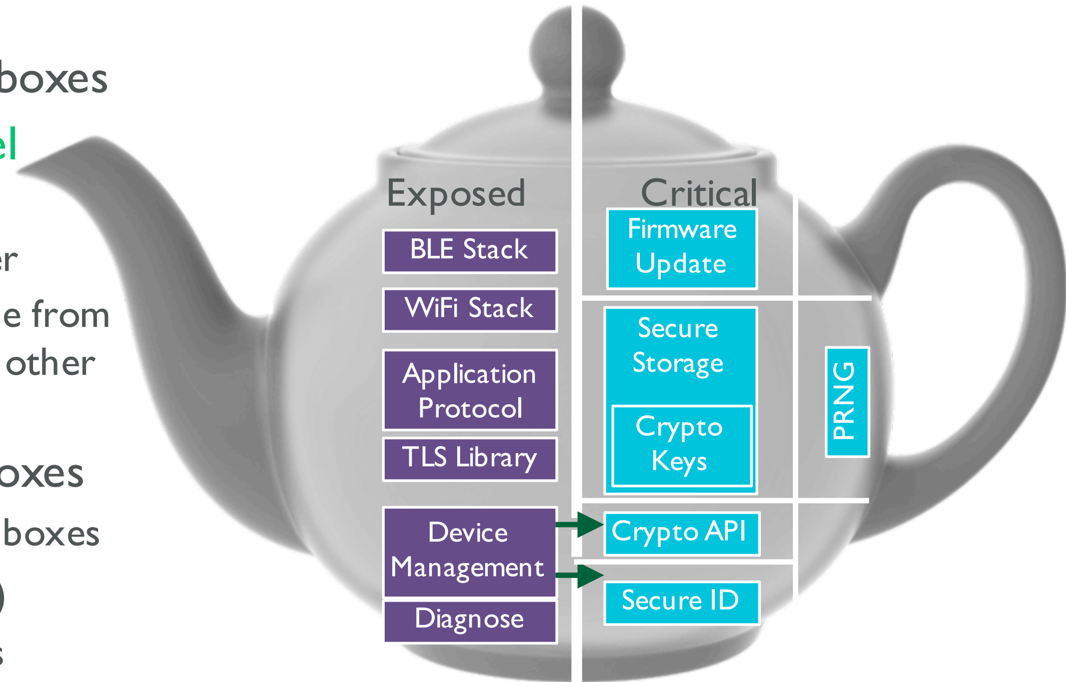
- Modules on the critical side require strong security coding practices
- Critical code rarely changes
- Exposed code developed rapidly
  - Faster time to market
  - Quick innovation cycles for exposed boxes
  - Still a secure product
  - Simple recovery from programming bugs in exposed code using secure boxes



© ARM 2015
ARM

The uVisor Design Principles

- Ease of use
- Hardware-enforced security sandboxes
- 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
- Per-box access control lists (ACL)
  - Restrict access to selected peripherals
  - Shared memories for box-box communication

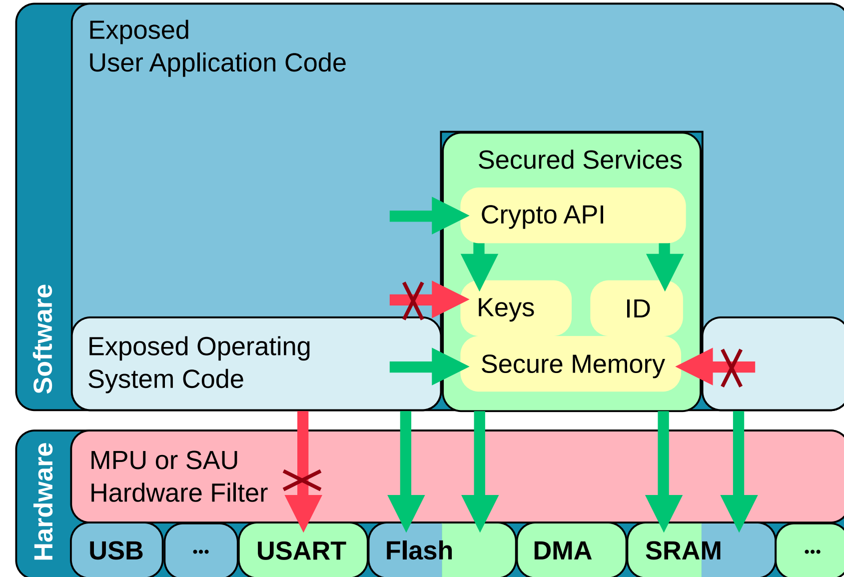


Future Versions of mbed OS will separate critical components like firmware update or raw cryptographic keys functions to ensure bugs in one secure module won't affect other secure modules.

© ARM 2015

The uVisor Boot Process on the ARMv7-M Architecture

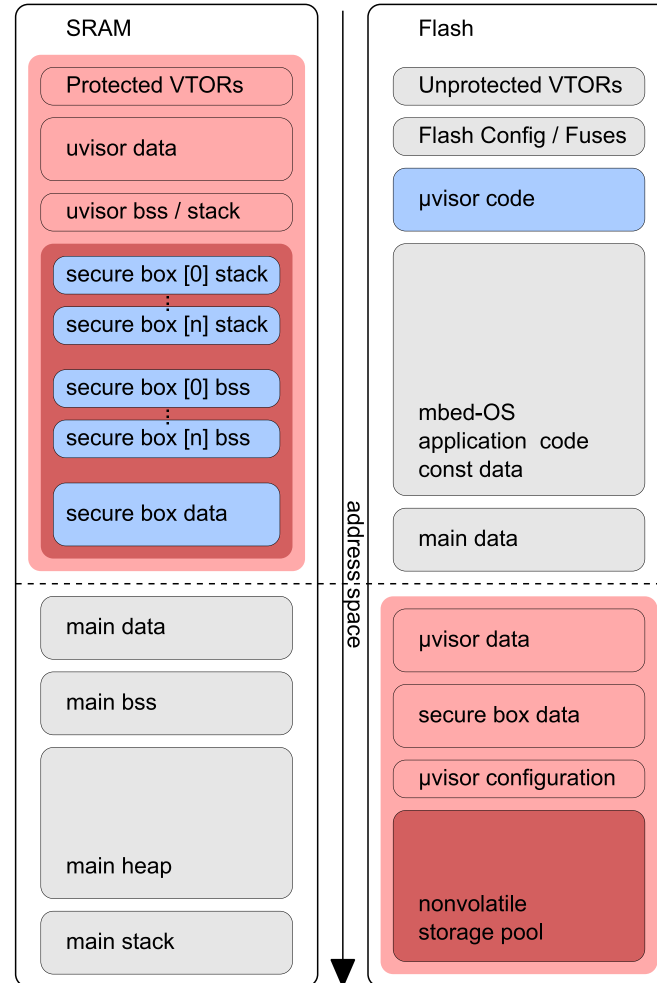
- uVisor initialized first in boot process
  - Private stack and data sections
  - Private data sections in flash for storing secrets
- Relocation of interrupts vector table into secure memory
- Initialization of memory protection unit based on box ACL's
  - Whitelist approach – only necessary peripherals are accessible to each box
  - Each box has private .bss data and stack sections
- De-privilege execution, continue boot unprivileged to initialize C/C++ libraries



© ARM 2015
ARM

The uVisor Memory Model

- uVisor allocates protected per-box stacks and detects under-/overflows during operation
- Main box memory accessible to all boxes
- All remaining per-Box data sections are protected by default:
  - Secure Per-Box Context Memory
  - Shared data/peripherals with other boxes on demand
  - uVisor resolves ACLs during boot and identifies ACL collisions
- uVisor code sections visible to everybody
- Empty flash memory is made available to the system as configuration storage – write access only through configuration API



© ARM 2015
ARM

“Developers, developers,
…, developers”

– Steve Ballmer

 © ARM 2015

ARM

Enable uVisor and Share Peripherals Across Boxes

```
#include <uvisor-lib/uvisor-lib.h>

/* set uvisor mode (enable) */
UVISOR_SET_MODE(UVISOR_ENABLED);
```

- enable uVisor – optionally with shared peripherals across boxes:

```
#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},
};
/* set uvisor mode (enable) */
UVISOR_SET_MODE_ACL(UVISOR_ENABLED, g_background_acl);
```

 © ARM 2015

Setting up Protected Sandboxes

```
/* create 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},    /* a peripheral */
};


/*required stack size */
#define BOX_STACK_SIZE 0x100

/* configure secure box compartment */
UVISOR_BOX_CONFIG(my_box_name, g_box_acl, BOX_STACK_SIZE, BoxContext);
```

 © ARM 2015

ARM

Define Function Call Gateways Across Security Boundaries

```
uint32_t secure_gateway(box_name, uint32_t target_fn, ...)/*pre-processor macro*/
```

- Call box functions using the security context of the target box ("sudo")
- Secure call gateways support up to four arguments and a return value

```
/* the box is configured here */
...

/* the actual function */
extern "C" uint32_t __secure_sum(uint32_t op1, uint32_t op2)
{
    return op1 + op2;
}
/* the gateway to the secure function */
uint32_t secure_sum(uint32_t op1, uint32_t op2)
{
    return secure_gateway(my_box_name, __secure_sum, op1, op2)
}
```


© ARM 2015

Secure uVisor API

- Accelerated secure peripheral access API with bit-level ACL's
  - Security verified during installation on device or before firmware signing on the server
  - Attackers can't create new peripheral access calls without writing to Flash
- IRQ API for unprivileged interrupt handling
  - Interrupts are executed unprivileged in the box security context of the owning box
  - CPU registers cleared for protecting the interrupted box against data leakage

```
void      vIRQ_SetVector(uint32_t irqn, uint32_t vector);
uint32_t  vIRQ_GetVector(uint32_t irqn);
void      vIRQ_EnableIRQ(uint32_t irqn);
void      vIRQ_DisableIRQ(uint32_t irqn);
void      vIRQ_ClearPendingIRQ(uint32_t irqn);
void      vIRQ_SetPendingIRQ(uint32_t irqn);
uint32_t  vIRQ_GetPendingIRQ(uint32_t irqn);
void      vIRQ_SetPriority(uint32_t irqn, uint32_t priority);
uint32_t  vIRQ_GetPriority(uint32_t irqn);
```


© ARM 2015

Secure uVisor Interrupt API

- Interrupt ownership is exclusive – multiple boxes cannot register for the same interrupt
- Registration based on first-come-first-serve: sharing IRQ's only possible via box API services
- uVisor remembers association between Interrupt handler and box security context during registration: other boxes can't access that interrupt till its released

```
#define MY_HANDLER_IRQn  42
void my_irq_handler(void)
{
    ...
}
int register_my_irq_handler(void)
{
    vIRQ_SetVector(MY_HANDLER_IRQn,  (uint32_t)  &my_irq_handler);
    vIRQ_EnableIRQ(MY_HANDLER_IRQn);
}
```


© ARM 2015

“The Devil is in the details, but so is salvation”

– Hyman G. Rickover, United States Navy Admiral

© ARM 2015

Call Gateway Internals for ARMv7-M Systems

- 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;
})
```


© ARM 2015
ARM

uVisor Outlook – Existing ARMv7-M vs. TrustZone for ARMv8M

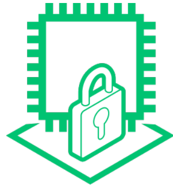
- ARM mbed-OS uVisor security model of TrustZone for ARMv8M remains the same as the ARMv7-M security model
- TrustZone for ARMv8M enables bus level protection in hardware
  - ARMv7-M requires software API filters for DMA access and other security critical operations
  - ARMv8-M can filter for DMA access for requests initiated by unprivileged code on bus level
- TrustZone for ARMv8M 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

© ARM 2015
ARM

mbed OS: Architected Security from the Ground Up

Lifecycle Security
mbed OS Secure Identity, Config and Update

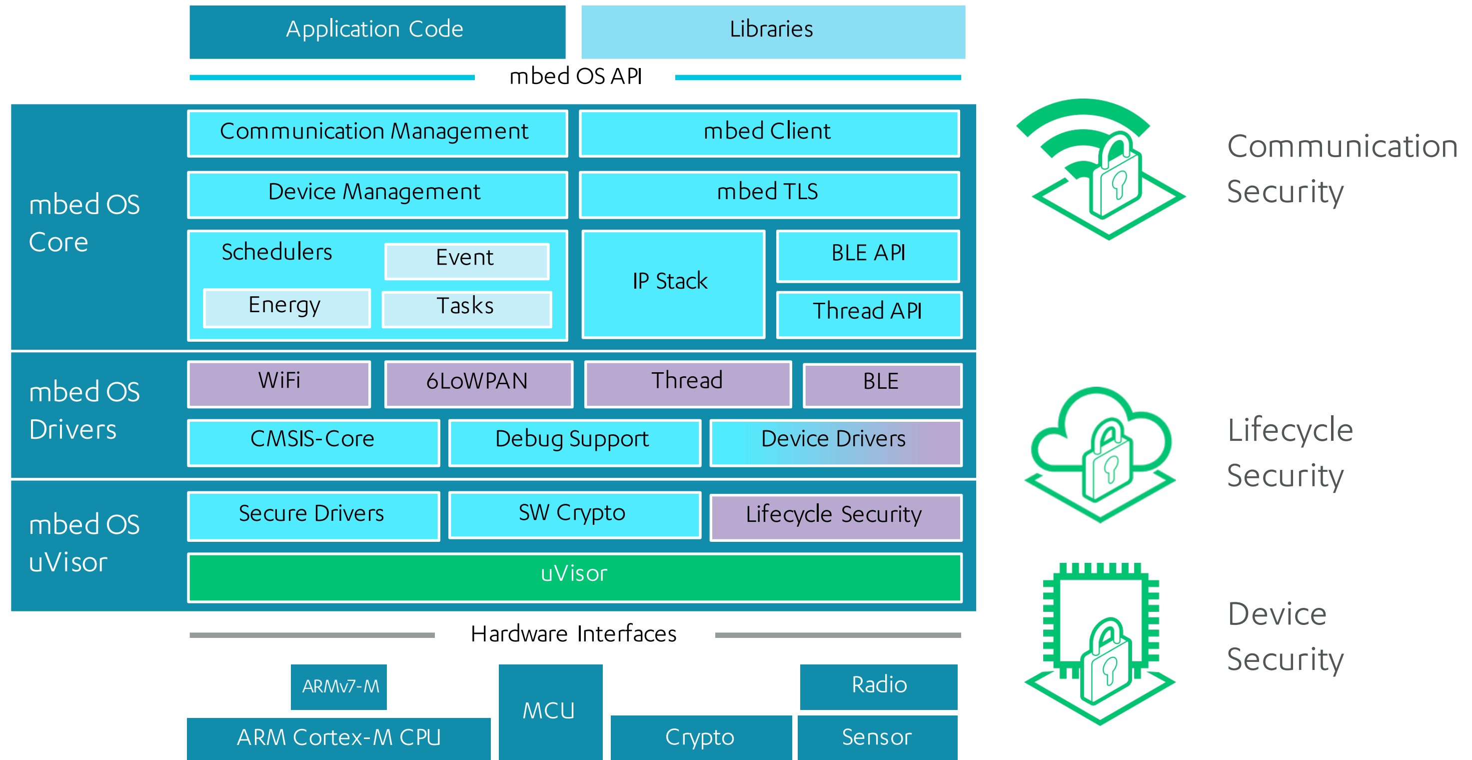
Communication Security
mbed TLS



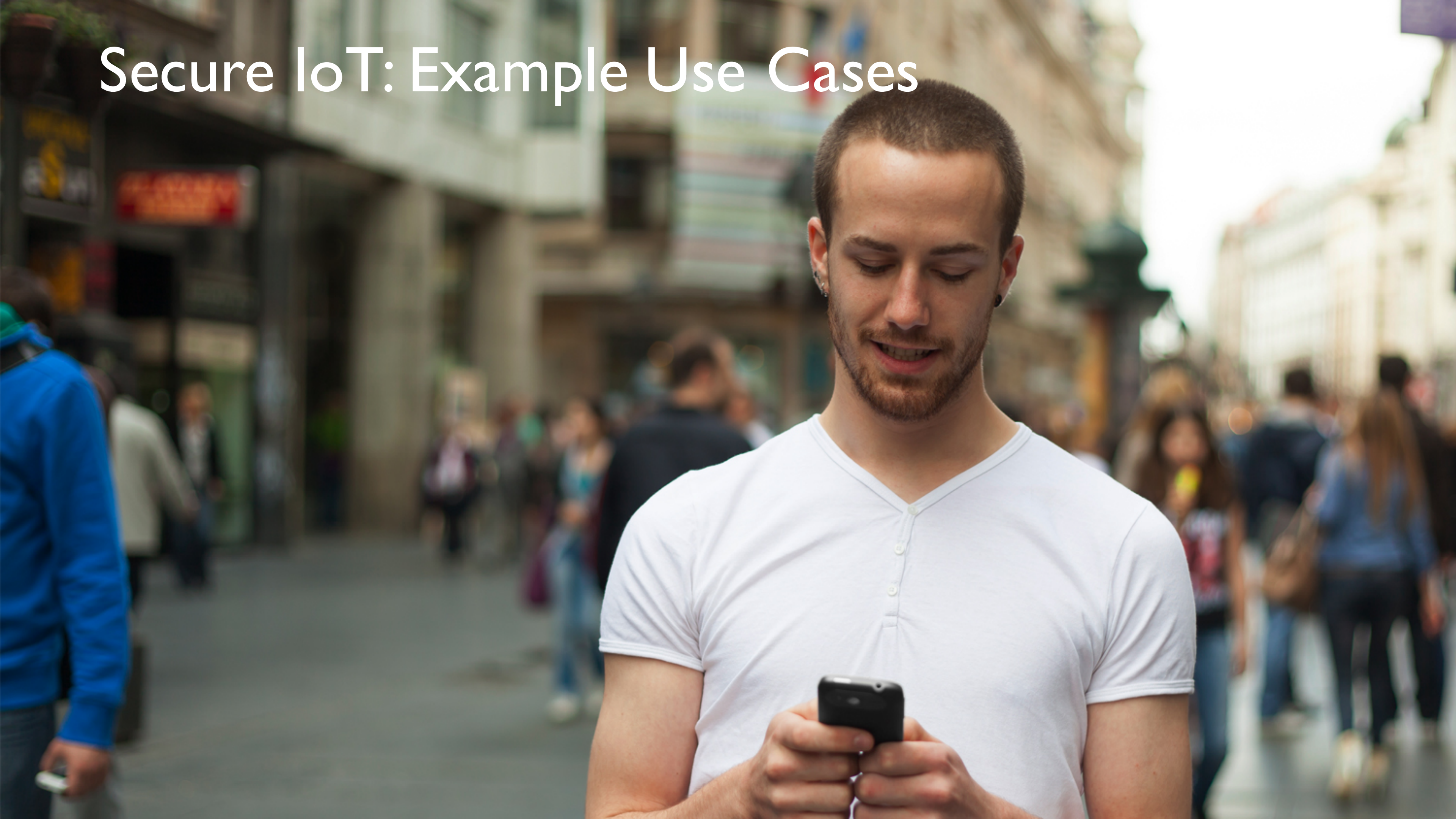
Secure Code Compartments
mbed OS uVisor on ARMv7-M MPU

© ARM 2015
ARM

mbed OS: Architected Security from the Ground Up

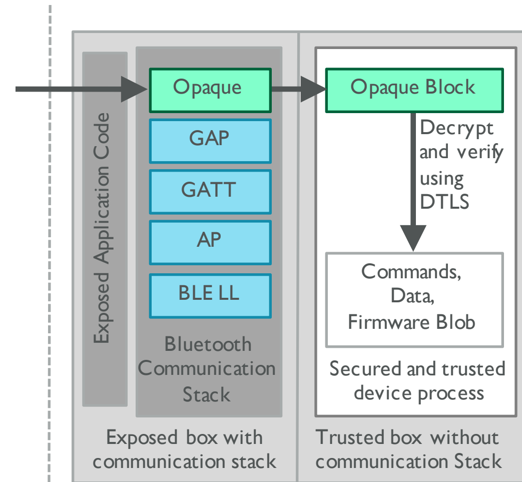


© ARM 2015


Secure IoT: Example Use Cases

Case Study: Secure Server Communication

- Trusted messages contain commands, data or firmware parts
- Box security not affected by communication stack exploits or infections outside of trusted box
- Payload delivery is agnostic of protocol stack
- Resilient box communication over the available channels
  - Ethernet, CAN-Bus, USB, Serial
  - Bluetooth, Wi-Fi, ZigBee, 6LoWPAN

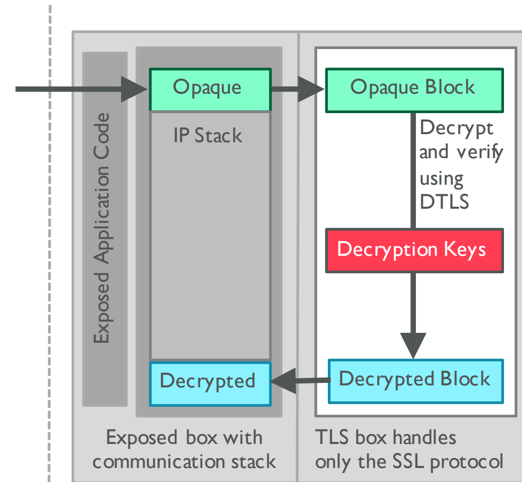


IoT Device owned by user.
Initial identity provisioned by System Integrator
Messages delivered agnostic of communication stack

© ARM 2015

Case Study: Secure Server Communication

- Communication protected by mbed TLS
- Raw message payloads decrypted and verified directly by protected code
  - mbed TLS box not exposed to communication protocol stack bugs
  - No interference by other boxes
  - Low attack surface
- Authentication and encryption keys are protected against malware
- Malware can't interfere without knowing the encryption or signing keys

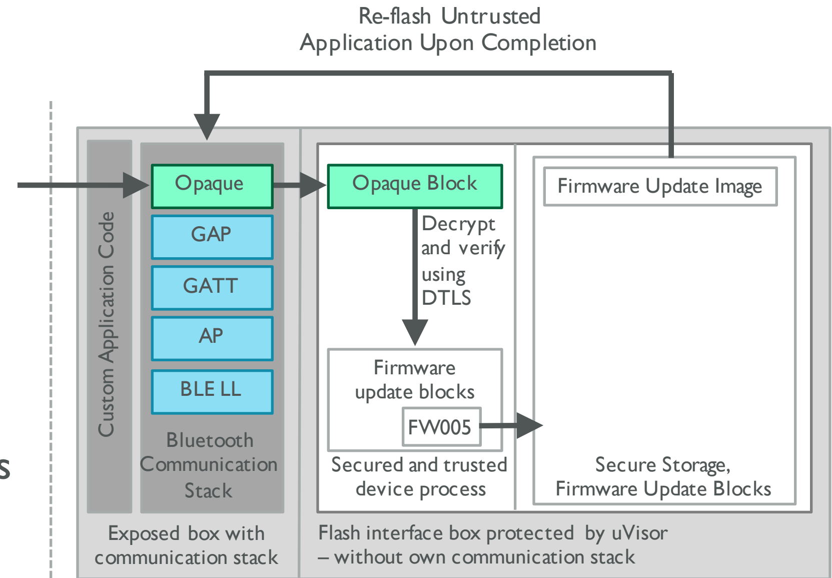


Initial keys provisioned by System Integrator. Messages decoded independent of stacks using mbed TLS in separate security context

© ARM 2015

Case Study: Secure Remote Firmware Updates

- Firmware manifest block augments existing firmware formats with safety and security features
  - Downgrade attacks prevented
  - Devices or device classes directly addressed by updates to improve safety
  - Efficient firmware distribution in mesh networks
  - Support for partial & chained firmware updates
- Crypto watchdog box enforces remote updates even for infected devices
  - New firmware applied after all blocks are received, decrypted and verified
  - Trusted side updates itself using a boot loader



© ARM 2015
ARM

Case Study: Controlled Malware Recovery

- Secure box detects malware infection
  - Enforces communication through the exposed side to the server
  - Receives latest security rules and virus behaviour fingerprints for detection
  - Shares detected pattern fingerprint matches with control server
  - Distributed detection of viruses and live infrastructure attacks
- When communication with the server breaks for a minimum time, parts of the device stack are reset to a known-good state
  - Prevents malware from staying on the device
  - Switches to a safe mode to detect network problems or to remotely update the firmware

© ARM 2015
ARM

"The mantra of any good security engineer is: "Security is not a product, but a process" It's more than designing strong cryptography into a system; it's designing the entire system such that all security measures, including cryptography, work together."

– Bruce Schneier

 © ARM 2015

ARM

Thank You – Questions?

Get the latest information on mbed security…

https://www.mbed.com/en/technologies/security/

…. and follow uVisor development on github:

https://github.com/ARMmbed/uvisor/

(uVisor is shared under Apache 2.0 license)

ARM

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.
All other marks featured may be trademarks of their respective owners.

Copyright © 2015 ARM Limited