ARES'13
Regensburg, Germany

Armored

CPU-bound Encryption for Android-driven ARM Devices

Johannes Götzfried, Tilo Müller

Department of Computer Science
Friedrich-Alexander University of Erlangen-Nuremberg

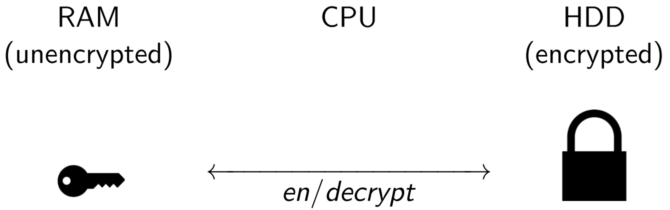
September 4, 2013

Full Disk Encryption

- Full disk encryption (FDE) protects data against *physical lost* and theft of the hard drive
- It does not protect against remote attacks

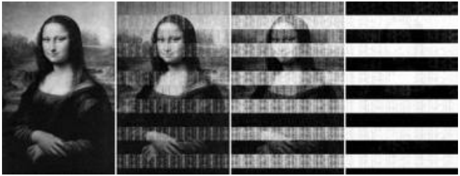
Software Disk Encryption

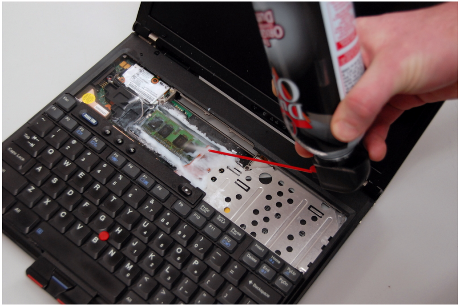
Current (software-based) FDE solutions do *not* protect data effectively if an adversary gains *physical access*!



Coldboot Attack

Disk Encryption Key in RAM
→ Exploit remanence effect of RAM





Encryption on Android

Encryption on Android

- Since Android 4.0 aka Ice Cream Sandwich (ICS)
- Based on dm-crypt (device-mapper and Linux Crypto API)
- Only the user partition `/data` is encrypted
- Mode `aes-cbc-essiv:sha256` is enforced with 128-bit keys

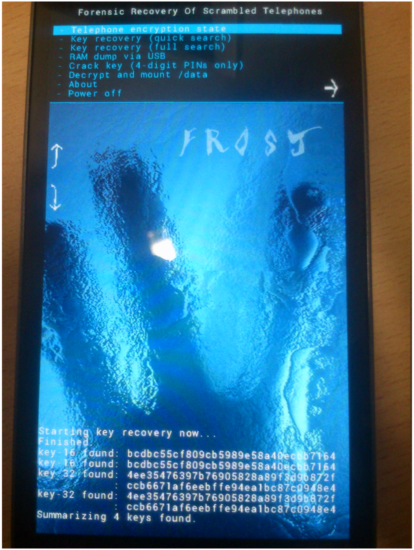
Of course encryption is possible with all common Linux distributions that run on ARM, too!

Coldboot Attack with Smartphones

FROST: Forensic Recovery Of Scrambled Telephones

And it works with smartphones too!

In this example: Galaxy Nexus



Attacks on Main Memory

Memory attacks require target systems to be *running* or *suspended*:

- Lost and theft of suspended laptops
- Confiscation of running servers
- But smartphones are **always on**

Basically *all* memory contents can be read out
→ We focus on the security of **disk encryption keys**!

ARMORED Security Policy

On **ARM** we **O**bstruct the **R**ecovery of **E**ncryption Keys from **D**RAM:

- AES implementation solely on the ARM microprocessor
- No sensitive information enters RAM
  - secret keys
  - key schedule
  - all intermediate states
- Only processor registers are used as storage

It has already been done ...

... on x86:

- FrozenCache
- LoopAmnesia
- TRESOR
- TreVisor

TRESOR

- uses the x86 debug registers *dr0* to *dr3* as key storage
- utilizes SSE registers to execute the AES algorithm
- implements AES using AES-NI

But ARMORED is the first CPU-bound encryption for ARM devices!

Challenge

Security Policy

No valuable information about the AES key or state should be visible in RAM at any time

→ Implement AES without using RAM at all

- no runtime variables
- no stack
- no heap

→ ARMORED core is written in pure ARM assembler
→ Misuse registers as key storage

Key Storage

Mix of breakpoint and watchpoint registers:

- Only accessible from kernel space
- seldom used by end-users

Memory alignment

- instructions are 4 bytes long and 4 bytes aligned
- two least significant bits of break- and watchpoint registers are zero

→ divide key-sequence into 16 bit chunks (for simplicity)

- store parts to the 16 most significant bits of the registers
- 4 breakpoint and 4 watchpoint registers: 128 bit
- PandaBoard: even 6 breakpoint and 4 watchpoint registers

→ AES-128 is possible, enough for Android's disk encryption

Working Register Set

NEON register set as temporary working storage:

- SIMD instruction set
- supported by many chips, e.g. Cortex-A9
- sixteen 128-bit registers, i.e. 256 bytes
- 64-bit and 128-bit SIMD operations
- access on byte granularity

Example

```
/* register defs */
rstate  .qn q0
rhelp   .qn q1
rk1     .qn q2
rk1d0   .dn d4
rk1d1   .dn d5
```

```
/* xor sbox(key[index]) onto r2 */
.macro ks_box index base rk
  vmov.u8 r3, \rk\()d0[\index]
  ldr     r3, [\base, r3, lsl #2]
  eor     r2, r2, r3
.endm
```

Gladman's AES Method

TRESOR implementation relies heavily on AES-NI

- AES-128 consists of basically 10 times `aesenc`

ARM has no AES-NI instruction set
→ use Gladman's AES Method

- based on table lookups
- efficient without special hardware

Specialities with ARM assembler

- RISC: all instructions are 4 bytes
- 4-byte base address of table cannot be loaded as immediate value
- manually generate constant pool and store pool address to register
- get base address register indirect

Key Schedule

Conventional AES:

- round keys are calculated *once* and then stored in RAM (for performance reasons)

ARMORED:

- on-the-fly round key generation (entire key schedule is too big to be stored inside the CPU)

Example

```
.macro key_schedule
  eor          r1, r1, r1
  ldr          r7, [r12]
  add          r8, r7, #1024
  add          r9, r8, #1024
  add          r10, r9, #1024
  ldr          r11, [r12, #4]
  generate_rk  rk1, rk1
  generate_rk  rk1, rk2
  generate_rk  rk2, rk3
  generate_rk  rk3, rk4
  generate_rk  rk4, rk5
  generate_rk  rk5, rk6
  generate_rk  rk6, rk7
  generate_rk  rk7, rk8
  generate_rk  rk8, rk9
  generate_rk  rk9, rk10
.endm
```

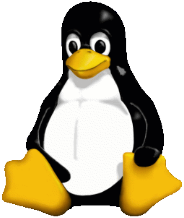
Kernel Patch

ARMORED is designed as a Linux kernel patch for three reasons:

1. dm-crypt and Android FDE uses the Linux Crypto API
2. Problem: unprivileged user access to debug registers
   $\rightarrow$ Solution: patch `ptrace` syscall
3. Problem: swapping of registers due to context switches
   $\rightarrow$ Solution: introduce *atomicity*

ARMORED is implemented in kernel space (currently Linux 3.2)

$\approx$ 1700 LOC
$\approx$ 500 lines assembly code



Atomic Sections

- OS regularly performs context switches
- if ARMORED is active this context comprises sensitive data
  → run ARMORED atomically (per 128-bit input block)

Example

```
void encrypt(struct crypto_tfm *tfm,
             u8 *dst, const u8 *src)
{
  unsigned long irq_flags;

  preempt_disable();
  local_irq_save(irq_flags);

  encblk_128(dst, src);

  local_irq_restore(irq_flags);
  preempt_enable();
}
```

Development Platform

Main development and testing was done on a PandaBoard running with Ubuntu 12.04 LTS (Precise Pangolin)

A Galaxy Nexus running Android 4.0 (Ice Cream Sandwich) has been tested as well

Security

ARMORED: nothing but the output block is written *actively* to RAM

But: sensitive data may be copied into RAM *passively* by OS side effects (e.g. interrupt handling, scheduling, swapping, etc.)
→ observe RAM of a ARMORED system at runtime

Tests

- Use FROST to actually perform a coldboot attack
- Look for keyschedule in RAM using *AESKeyFind*
- Look for the key in RAM (search for longest match)

Physical RAM was dumped using *LiME* – the Linux Memory Extractor
→ We did not find the key (longest match was 4 bytes)

Correctness

How to ensure that our implementation is correct?

- Linux kernel provides a test manager
    - check with official AES test vectors
- Encrypt random data with ARMORED and decrypt with generic AES
- Encrypt random data with generic AES and decrypt with ARMORED

→ We have good evidence that our implementation is correct

Performance

At first ARMORED was 4.5 times slower than generic AES

Improvement: Larger atomic sections

- Process more input blocks per atomic section
  - reduce number of necessary key schedules
- How many blocks per section?
  - interactivity is no problem (1-2 microseconds vs. milliseconds)
  - could make sections large (up to 1024 blocks)
  - but: only 512 bytes per sector, i.e. maximal 32 blocks
- Necessary to change modes of operation: ECB, CBC, CTR

→ two ARMORED variants: 16 blocks or only 1 block per section

Performance Results

Reading 400 MB random data from encrypted RAM disk:

- Generic AES: 15.55 MB/s
- ARMORED 1x: 3.57 MB/s
- ARMORED 16x: 6.76 MB/s

Comparison of coldboot resistant implementations:

	slowdown
TRESOR	1.5
TreVisor	1.5
LoopAmnesia	2.0
ARMORED 1x	4.5
ARMORED 16x	2.3

Limitations

Or why do we call it proof of concept?

Installation of ARMORED on smartphones is not very easy

- A kernel patch is not a user friendly application
- You might even do not have the code or parts of it

Bootstrapping problems

- How to get the key into the debug registers?
- Currently via adb and a sysfs interface

Integration into the android boot prompt

- Would be easily possible
- Just change hardcoded cipher and use sysfs interface

Confidentiality

Almost impossible to ensure that no password or key fragments remain within RAM

Conclusion

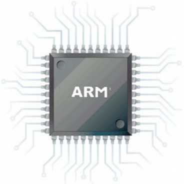
ARMORED withstands coldboot attacks and protects your DEK

It does not prevent:

- Local privilege escalation
- JTAG attacks
- Loss of other sensitive data in RAM

ARMORED can be used

- practical on ARM based laptops
- on smartphones only as proof of concept



ARMORED is the first CPU-bound encryption for ARM devices

Thank you for your attention!

Further Information:

- http://www1.cs.fau.de/armored