ASIACCS'16
Xi'an, Shaanxi, China

RamCrypt: Kernel-based Address Space Encryption for User-mode Processes

Johannes Götzfried*, Tilo Müller*, Gabor Drescher*,
Stefan Nürnberger†, and Michael Backes†

*Department of Computer Science
FAU Erlangen-Nuremberg, Germany

† Center for IT-Security, Privacy, and Accountability (CISPA)
Saarland University, Germany

June 3, 2016

Memory Disclosure

RAM contains lots of sensitive data:

- User passwords or login credentials
- Cryptographic keys
- Personal data and credit card information

→ Information is only protected by logical means, e.g., by the OS

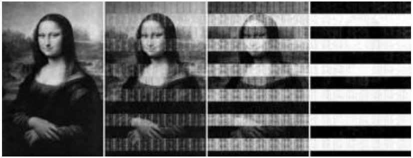
Sources of inadvertent memory disclosures:

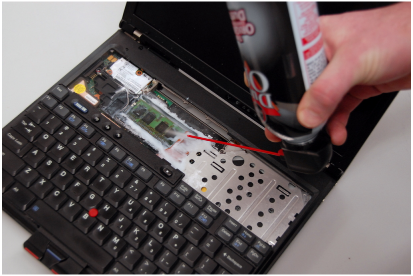
- Swap files and crash reports (core dumps)
- Vulnerable kernel drivers / kernel drivers with backdoor
  Example: Samsung's firmware for the Exynos chipset offered an unprotected `/dev/mem` device

Physical Memory Disclosure

Physical Attacks on RAM:

- By using DMA
  Example: Firewire
- Cold Boot Attacks





Data Lifetime

Goal: Reducing data lifetime of sensitive information within RAM:

- Requires data lifetime knowledge
- Traditional wiping approaches fail (no transparency)

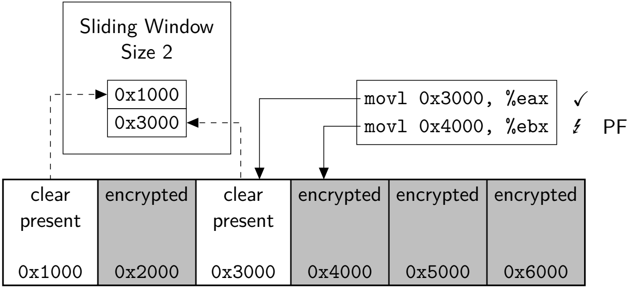
→ Transparent data encryption effectively hides information

RamCrypt: Idea

Transparently encrypt data within process address spaces:

- On a per-page basis
- Only encrypt data (anonymous private mappings)
- Only a small set of pages remains unencrypted

Sliding window instead of only single page:



→ Sliding window size is a configurable security parameter

RamCrypt: Background

Prototype implementation as a Linux kernel patch:

- Builds upon the Linux kernel patch TRESOR
- CPU-bound implementation of AES
- Stores the key and all intermediate values in CPU registers

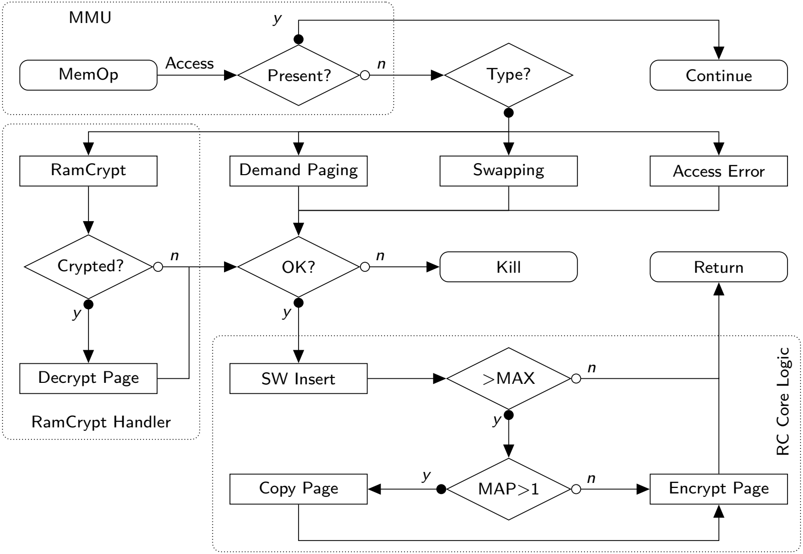
→ No cryptographic keys or key material ever enter RAM

Linux virtual memory management:

- Page faults are used to handle everything
- Highly relies on demand paging
- Copy-on-Write (COW) during forking

→ Implement RamCrypt in the page fault handler of Linux

RamCrypt: Workflow



RamCrypt: Managing Memory Pages

Catching accesses to encrypted memory pages:

- Clear the present flag (bit 0) to cause page faults
- Set a new flag (bit 10) indicating that the page is encrypted
- Second software defined flag (SW2) is available for PTEs

63		11	10	9	8	7	6	5	4	3	2	1	0
NX	· · · Page Table Address · · ·	SW3 HID	SW2 SPL	SW1 SPC	G	S	0	A	C	W	U	R	P

63		11	10	9	8	7	6	5	4	3	2	1	0
NX	· · · Physical Page Address · · ·	SW3 HID	SW2 RC	SW1 SPC	G	0	D	A	C	W	U	R	P

Available to the OS

In addition: One flag within physical page's management structure

- Needed to handle COW semantics

RamCrypt: Multithreading and Address Space Creation

Multithreading support:

- RamCrypt is fully compatible with multithreaded applications
- Sliding Window size is per process not per thread
- Possible to give fixed guarantees

→ Performance suffers from too many threads

Support for forking:

- Forking is the way of creating a new process in Linux
- PTEs and the sliding window are copied during `fork()`
- Only PTE of current process is modified during page fault
- Flag within physical structure is used to check whether decryption is really necessary
- Multiply mapped pages are copied before being encrypted by core logic

RamCrypt: Loading of a Binary

RamCrypt is enabled on a per-process basis:

- Binaries need to be flagged
- RamCrypt reuses the `PT_GNU_STACK` program header of an ELF executable

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
...													RC	R	W	X

- User-mode utility for flagging binaries is provided

Loading of a flagged binary:

- RC bit is checked for during `execve()` system call
- The address spaces of the process and all child processes are encrypted (RC bit is inherited during `fork()`)
- Executing a binary with RC bit unset disables encryption

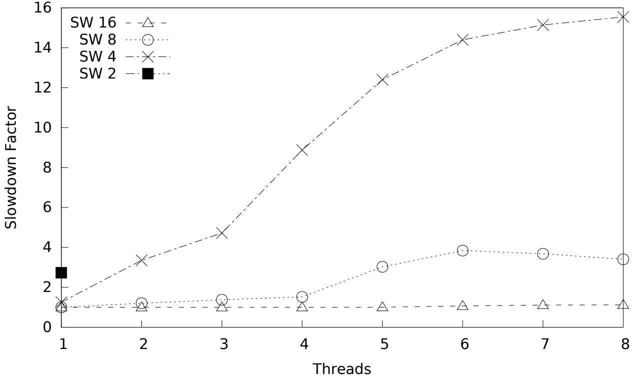
RamCrypt: Cipher

TRESOR (CPU-bound implementation of AES):

- Configured to behave like AES-128 in XEX mode of operation
- IV to build tweak: `vaddr` || PID
- Supports page relocation (but no shared pages)
- Using PIDs prevents attackers from guessing page contents
- After `fork()`: PID of the parent is used until call to `execve()`

RamCrypt: Sliding Window Performance Impact

Overhead of RamCrypt-enabled benchmark (*sysbench*):



- For a SW size of sixteen, our implementation scales (12% slowdown with eight threads)
- Singlethreaded run with SW size two: 170% slowdown
- Singlethreaded run with SW size four: 25% slowdown

RamCrypt: Practical Security Analysis

RamCrypt-enabled *ngnix* webserver delivering SSL-encrypted HTML pages under maximum load:

		Temporal Exposure per Page (%)		
		n=4	n=8	n=16
Secret Key Pages		3.07	14.37	21.68
All Pages	Min	0.0000	0.0005	0.0017
	Avg	7.63	12.66	17.95
	Max	99.83	99.76	99.99
	StdDev	19.77	21.82	25.43

→ Default SW size four: 3% exposure time for secret key pages

Conclusion

Limitations:

- Kernel or driver buffers are not protected by RamCrypt
- RamCrypt cannot protect against attacks such as Heartbleed
- Noticeable performance drawback for multi-threaded programs

RamCrypt protects data of whole process address spaces:

- Effectively protects against physical memory disclosure attacks
- Can be enabled on a per-process basis without recompilation
- Only 25% slowdown for single-threaded processes with a sliding window size of four

Thank you for your attention!

Further Information:

https://www1.cs.fau.de/ramcrypt