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

MMU

MemOp

Access

Present?

Type?

Continue

Demand Paging

Swapping

Access Error

RamCrypt

Crypted?

OK?

Kill

Return

RamCrypt Handler

Decrypt Page

SW Insert

> MAX

Copy Page

MAP > 1

Encrypt Page

RC Core Logic
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

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

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RC</td>
<td>R</td>
</tr>
</tbody>
</table>
```

- 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
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:

<table>
<thead>
<tr>
<th>Temporal Exposure per Page (%)</th>
<th>n=4</th>
<th>n=8</th>
<th>n=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret Key Pages</td>
<td>3.07</td>
<td>14.37</td>
<td>21.68</td>
</tr>
<tr>
<td>Min</td>
<td>0.0000</td>
<td>0.0005</td>
<td>0.0017</td>
</tr>
<tr>
<td>Avg</td>
<td>7.63</td>
<td>12.66</td>
<td>17.95</td>
</tr>
<tr>
<td>Max</td>
<td>99.83</td>
<td>99.76</td>
<td>99.99</td>
</tr>
<tr>
<td>StdDev</td>
<td>19.77</td>
<td>21.82</td>
<td>25.43</td>
</tr>
</tbody>
</table>

→ 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