Hardened Anti-Reverse Engineering System

@JacobTorrey

Assured Information Security (@ainfosec)

March 2015
Who am I?

- Advising Research Engineer at Assured Information Security (words are my own)
- Site lead for Denver, CO office
- Leads low-level Computer Architectures research group
- Plays in Intel privilege rings $\leq 0$
- Ultra-runner, ultra-cyclist & mountaineer
Overview

• HARES provides the ability to execute *fully-encrypted* binaries
• Minimal performance impact at ~2% on unmodified Intel Core-i series processor
• Prevents key or instruction leakage even to compromised OS kernel
• Can protect *unmodified* binary applications *without* source or recompilation
Problem Statement

- Application code can be used to develop attacks
- Algorithms exposed to copying or theft
- Code can be reused for unintended purposes (ROP)
Current State-of-the-Art

- Obfuscation and packers — Analysis tools and live debugging can recover instructions
- VM-based obfuscation — Can still be mapped to x86 and impacts performance
- Encrypting entire OS with trusted boot — Only prevents against offline attacks
AES-NI

- In response to software-based caching attacks on AES, Intel released instruction set to support AES
- Hardware logic is faster, and more protected
- Supports 128-bit and 256-bit AES
- Provides primitives, still requires engineering to make a safe system on top of these
TRESOR

- Uses AES-NI and CPU debug registers to provide accelerated, cold-RAM resistant AES on Linux
- Key loaded early boot
- Kernel patch to prevent applications reading debug registers
TLB-Splitting

- Translation lookaside buffer (TLB) is a cache for virtual — physical address translations
- Used to improve paging performance
- Logically treated as single entity, *physically multiple components*
- Switches x86 platform from apparent Von Neumann to Harvard architecture:
  - Used by PaX/GRSecurity to emulate no-execute bit
  - Shadow Walker used for memory-hiding root-kit functionality
TLB-Splitting (cont.)

**Figure 1:** TLB during normal operation

**Figure 2:** Split TLB
With Nehalem micro-architecture, an L2 cache was introduced, the S-TLB; breaks split-TLB assumptions.
MoRE: Measurement of Running Executables

- DARPA Cyber Fast Track program
- Explored using TLB-splitting for measurement/integrity verification of interleaved application
  - Immutable code page (can repeatably measure in real-time)
  - Mutable data page (for variable isolation)
- Used EPT granular permissions to simulate a split-TLB on newer CPUs with S-TLB
- Thin-VMM to simulate Harvard architecture on per-process basis
VMX Thin-Hypervisor

- Loaded as Windows 7 kernel driver
- Based on vmcpu root-kit example
- No emulation of devices, OS retains direct access to HW
- Minimal performance impact
- Can use VMCS exit conditions to track certain architectural event
On-CPU AES

- “Hoists” TRESOR on-CPU AES into VMM
- Adds VMCS exit condition for debug register accesses (return NULL or silently discard write)
- Decrypts executable sections of program into execute-only memory
Process Creation Monitoring

• Registers call-back for process creation
• Notified before execution and during termination
• Parses PE and identifies regions to decrypt and perform TLB-split on
• Uses hyper-calls to begin or terminate VMM support
TLB-Splitting

- All data-fetch requests, even from application itself, directed via EPT/TLB to encrypted page
- All instruction fetches are directed to execute-only, decrypted, memory
- Must track Windows application memory management events (COW) and ensure EPT structures correspond with OS-level structures
Test Cases

- Calculation of $\pi$ — Purposely inefficient power-series algorithm to approximate $\pi$
- Random Sort — Randomized CPU and memory access patterns to test the performance for non-consecutive cache-line accesses
- Coin-flip — Called many shared library functions to ensure compatibility
- Timers — Monitors performance impact as ratio of cycles to ‘wall’ time
Test Results

Overall, average performance impact was ~2%

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Average Execution Time (s)</th>
<th>Average HARES Execution Time (s)</th>
<th>Performance Impact (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pi</td>
<td>28.173</td>
<td>28.600</td>
<td>1.515</td>
</tr>
<tr>
<td>Randomized Sort¹</td>
<td>0.016</td>
<td>0.031</td>
<td>95.83</td>
</tr>
<tr>
<td>Coin Flip</td>
<td>1.778</td>
<td>1.809</td>
<td>1.762</td>
</tr>
<tr>
<td>Timers</td>
<td>15.013</td>
<td>15.023</td>
<td>0.069</td>
</tr>
</tbody>
</table>

Table 1: Performance Results for CLI Test-Suite

¹The randomized sort runs for such a short period that the initialization routine of the Windows process creation monitor almost doubles the execution time. If the TLB-splitting and periodic measurement is disabled, the run-time is still almost double. For this test, it is reasonable to assume that there is a constant increase of ~0.015s for each test case run under the VMX hypervisor, and due to this test case's short duration it skewed the percentage.
Tested Applications

- Aforementioned synthetic test-suite
- Microsoft Windows Notepad
- Microsoft Windows Paint
- Microsoft Windows Calculator

Usability of HARES-protected applications was not noticeably impacted to end-user
Demonstration System

Specifications for the demonstration system to overcome some prototype limitations:

- Intel Core-i series processor with AES-NI; Windows 7 32-bit OS
- Single processor [numproc=1] (Our thin-VMM only supports single core currently)
- 2GiB RAM [truncatememory=0x80000000] (Windows kernel memory layout changes > 2 GiB, and too lazy to update hard-coded values)
- No PAE/DEP [nx=AlwaysOff and pae=ForceDisable] (Again, hard-coded memory layout code)
Demonstration

[ALT-TAB]
Engineering Challenges

- Mixed code & data in PE section
- Paging out of application memory
- COW/relocation of application
Overcoming Challenges: Mixed Code & Data

- **Easily identifiable**
  - Import tables, debug tables, etc., are easily parsed and excluded from encryption

- **Not so easily identifiable**
  - Single purpose strings and other small data are often stored adjacent to the code that uses them and are difficult to identify in compiled code

- **Not a problem if source is available**
  - Compiler options can be used to create read-only sections

- **Binary only**
  - Reverse engineering — time consuming & unreliable
  - Provisioning/Learning Mode — it works for proof of concept, but unreliable for large programs
  - Debug symbols
Overcoming Challenges (cont.)

- Uses `MmProbeAndLockPages()` to prevent OS page-out — limited non-paged pool
- VM Exit on CR3 change to re-walk page-tables to detect COW — update TLB-split pages via hyper-call
Security Benefits

Protects against:

- Reverse-engineering and algorithmic IP theft
- “Weaponization” of crash case into RCE
- Mining application source for ROP gadgets
- Harvard architecture resistant to code injection attacks
Weaknesses

• JTAG/ICE/XDM
• Memory Dumping
• DMA
• SMM/AMT
• Side-channels
• Emulation/VMM
Overcoming Weaknesses & Future Work

- VT-d/IOMMU
- Cache-as-RAM
- DRTM launch (e.g., Intel TXT)
- Combining with unique compilation
Unintended Use-Cases

- Offensive key management is a less-studied practice and more challenging/likely to get wrong
- Easier to use for defense than offense
**AV Heuristics**

![VirusTotal](https://example.com/virustotal)

<table>
<thead>
<tr>
<th>SHA256</th>
<th>1acbe6931408a46c8f2d4f0d28c30a081641bde87ca15634a870b06c660a4b</th>
</tr>
</thead>
<tbody>
<tr>
<td>File name</td>
<td>notepad.exe</td>
</tr>
<tr>
<td>Detection ratio</td>
<td>4 / 57</td>
</tr>
<tr>
<td>Analysis date</td>
<td>2015-03-25 23:25:53 UTC (1 minute ago)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Antivirus</th>
<th>Result</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>Win32/Heur</td>
<td>20150326</td>
</tr>
<tr>
<td>Bkav</td>
<td>HW32.Packed.35C0</td>
<td>20150325</td>
</tr>
<tr>
<td>Rising</td>
<td>PE:Malware.XPACK-HIE/Heur1.9C48</td>
<td>20150325</td>
</tr>
<tr>
<td>Tencent</td>
<td>Trojan.Win32.YY.Gen.3</td>
<td>20150326</td>
</tr>
<tr>
<td>ALYac</td>
<td></td>
<td>20150326</td>
</tr>
</tbody>
</table>

- **However**, notepad.exe (unencrypted) with a 1-bit change to remove from MS white-list *also* hits 4/57
Concluding Remarks

- Demonstrates viability of encrypted execution on existing, common hardware
- Significantly increases reversing difficulty with minimal performance impact
- Provides vulnerable legacy systems “breathing room” until appropriate fixes can be implemented
- Intel SGX will be an exciting hardware extension to the platform and should be explored
Acknowledgments

• Mark Bridgman (@c0ercion) for his work on this effort
• Mudge (@dotMudge) & DARPA for supporting the precursor work (MoRE)
• Loc Nguyen (@nocsi_) & Ryan Stortz (@withzombies) for their input from a reverse engineering perspective
References

Formal references can be found in the whitepaper for:

- GRSecurity PAGEEXEC
- Shadow Walker
- Intel SDM
- TRESOR & TRESOR-Hunt
- ARIUM website
- Self-hashing applications
- CoreBoot Cache-as-RAM

and more.
Thanks!

See you all next year at Syscan 2016!
Questions & Discussion

- Thanks for your attention!
- Any questions? or let the heckling begin!
- Whitepaper can soon be found on my Twitter profile (@JacobTorrey: pinned-tweet)