real-time passive volatile memory inspection inside virtual machines

CanSecWest. march 2015

johnwwil[at]u.washington.edu
who am I?

• John Williams
  • security consultant @ Ernst & Young (emerging technologies)
  • embedded SW engineer (safety critical systems)
  • computer engineering @ University of Washington

• research interests
  • defensive security technologies
  • embedded security
  • trusted computing
talk roadmap

• motivations for research
• introduce memminer
• gaps in run-time protection
• steps to building a solution to address problem
• live demo of memminer
motivations for research

• looking at security of virtualized hosts
  • they’re everywhere: cloud, mobile, desktop, etc.
  • opportunity for different methods of protection

• memory-based rootkits on the rise
  • may never touch a disk
  • kernel of host cannot really be trusted

• trusting the data used for analysis
  • providing greater assurance
  • trusted hw is around the corner
what is memminer?

• agentless security indicator collection for virtual machines
  • monitors indicators that pose red flags in a system
  • provides standard output
  • logging of data over time for analysis

• somewhat hypervisor and guest system agnostic

• provides basis for executing response behaviors at VM level

• differs from existing solutions because it’s agentless
how does memminer work?

• memminer sits at hypervisor level and peeks into host using libvmi
• leverages rekall to track specific indicators that may be hidden from host (requires minimal knowledge of host)
• monitors indicators that may pose red flags in a system
  • (e.g. ssdt, apihooks, etc.)
• converts data to standard format (cybox)
• logs data over time into mongodb for independent tracking and longitudinal analysis
system attack surfaces

• modern operating systems have very high attack surface

• hypervisors have much smaller attack surface in comparison
  • much more trustworthy but neither can be fully trusted (i.e. hardened)
boot protections

• trusted boot for boot protections
  • Mechanisms for securing and authenticating the boot process

• examples
  • trusted execution extensions
  • UEFI
runtime protections

• “in-the-box” methodology is deficient
  • relies on kernel to protect itself
  • huge attack surface provides rootkits many options for specifically evading agents

• Ultimately is a cat-and-mouse game
  • example: PatchGuard bypassing
runtime protection gaps

• trusted boot doesn’t provide ongoing runtime protection

• complex code will have complex problems
  • e.g. third party driver loading

• what can we do to support the runtime protection of the guest?
gap scenarios considered

• controlled guest
  • has some sort of traditional agent
  • possibly has an anti-malware solution

• uncontrolled guest
  • no agent or other security controls
  • example: unmanaged VPS
functional goals

- support controlled guest
  - complement agent to enhance integrity
  - provide rudimentary rootkit detection
  - validate guest defenses

- monitor uncontrolled guest
  - develop of a truly agentless HIDS
  - provide indicators from within guest
  - look for modified structures
  - some of the things anti-malware does
tool landscape

• all perform in-the-box data collection
  • cuckoo
    • automated malware analysis
  • ossec
    • host-based intrusion detection
• google rapid response
  • incident response

• existing products support VMware
  • trend micro
outside guest protection

• “derivative delivery” is specialized
  • integration into hypervisor
  • interposition $\rightarrow$ modification/feedback

• “out-of-the-box” methodology
  • isolation $\rightarrow$ integrity
  • inspection $\rightarrow$ full visibility
out-of-the-box analysis

- benefits of properties
  - isolation provides integrity
  - Inspection provides full visibility
  - control $\rightarrow$ response

- implementation challenges
  - semantic gap
  - user/performance impact
  - what to do with data
solution components preview

• functional and user requirements
• memminer solution components
  • Libvmi + vmifs
  • Rekall
  • CyBox
user requirements

• no guest modification
• hypervisor independent
• no adverse performance impact
• outputs data in usable format
performance impact

• problem: introspect with minimal interference
  • how we access memory is important
  • full memory snapshots are costly
  • don’t want to pause the host
    • memory coherency

• solution: libvmi (virtual machine introspection)
  • direct memory access
  • shm-snapshot provides coherency
  • designed to work across different hypervisors
  • however, requires hypervisor support
  • also provides abstracted control

<table>
<thead>
<tr>
<th>hypervisor</th>
<th>uncontrolled guest</th>
<th>controlled guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>libvmi</td>
<td>✓ no guest modification ✓ hypervisor independent ✓ no adverse performance impact ❏ bridge semantic gap ❏ complement agent ❏ provide indicators ❏ outputs data in usable format</td>
<td></td>
</tr>
</tbody>
</table>

host analysis
bridging the semantic gap

• problem: bridging the OS semantic gap
  • inherent if we can’t query kernel
  • complex problem, system-dependent
  • we don’t want to have to tailor each install

• solution: forensic analysis tools
  • designed to bridge the semantic gap in raw memory dump
  • actively being developed

<table>
<thead>
<tr>
<th></th>
<th>uncontrolled guest</th>
<th>controlled guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>hypervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libvmi</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ✓ no guest modification
- ✓ hypervisor independent
- ✓ no adverse performance impact
- ✓ bridge semantic gap
- ❏ complement agent
- ❏ provide indicators
- ❏ outputs data in usable format

host analysis
forensics tools

• designed for malware hunting
  • random information dumping
  • lie detection (psxview)
  • raw socket detection
  • integrity checking (process dumping)
  • heuristic scanning, yara

• limitations
  • not designed for speed
  • not compact or hardened

• capable tools
  • volatility
  • rekall

---

<table>
<thead>
<tr>
<th>hypervisor</th>
<th>uncontrolled guest</th>
<th>controlled guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>libvmi</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>rekall</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- no guest modification
- hypervisor independent
- no adverse performance impact
- bridge semantic gap
- complement agent
- provide indicators
- outputs data in usable format
usable output

• problem: need to output structured data
  • malware red flags
    • ssdt, apihooks, etc.
  • operational system data
    • what’s normally collected by HIDS
    • longitudinal analysis

• solution: CybOX
  • Cyber observables expression
    • Objects and relationships (e.g. Process, File, DNS Cache)
    • [https://cybox.mitre.org/](https://cybox.mitre.org/)
  • standardized – composability

<table>
<thead>
<tr>
<th>uncontrolled guest</th>
<th>controlled guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>libvmi</td>
<td></td>
</tr>
<tr>
<td>rekall</td>
<td></td>
</tr>
<tr>
<td>CybOX</td>
<td></td>
</tr>
</tbody>
</table>

- no guest modification
- hypervisor independent
- no adverse performance impact
- bridge semantic gap
- complement agent
- provide indicators
- outputs data in usable format
integration opportunities

• Create an “agentless” agent
  • hypervisor-level agent representing host
  • ossec-hids
  • google rapid response

• support rootkit detection
  • signature-based
  • heuristic/behavioral based
  • cross-view based
  • integrity-based

<table>
<thead>
<tr>
<th>host analysis</th>
<th>uncontrolled guest</th>
<th>controlled guest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>libvmi</td>
<td>rekall</td>
</tr>
<tr>
<td></td>
<td>CybOX</td>
<td></td>
</tr>
</tbody>
</table>

• no guest modification
• hypervisor independent
• no adverse performance impact
• bridge semantic gap
• complement agent
• provide indicators
• outputs data in usable format
Demo

- windows 7 machine
- monitoring process/service lists, ssdt
- output data into mongodb
how can this be evaded

• timing attacks
  • presence of VM and can point toward monitoring

• functional evasion
  • i.e. detection and avoidance

• Functional modification
  • i.e. modifying target code
  • DKOM (direct kernel object modification)

• VMM exploitation, firmware exploitation

• attack the tools
limitations of memminer

- Increasing attack surface
- not incredibly lightweight
- some assembly required
- only gives you main memory
- limited by hypervisor support
  - xen, qemu+kvm supported by libvmi
benefits of memminer

• uses same tools that are being used for forensic analysis

• rekall is under active development

• many options for leveraging things
  • periodically integrity check your favorite process
  • pause machine on bad things
next steps

• better support for guest memory access APIs

• minimize solution

• integrate with something
what do we need ultimately

• semantic reconstruction tools
  • high-performance
  • lightweight
  • composable

• forensic analysis tools
  • easily scriptable
  • based on semantically reconstructed data

• virtual host-based IDS

• ultimately afforded
  • self-protection with isolation
  • self-healing through modification
Questions?

- https://goo.gl/LdItuD
- https://github.com/johnwwil/memminer

Contact me: johnwwil[at]u.washington.edu