Live and Trustworthy Forensic Analysis of Commodity Production Systems

Lorenzo Martignoni[1] Aristide Fattori[2] Roberto Paleari[2]
Lorenzo Cavallaro[3]

[1]University of California at Berkeley [2]Università degli Studi di Milano

[3]Vrije Universiteit Amsterdam

13th International Symposium on Recent Advances in Intrusion Detection
Ottawa, Ontario, Canada, September 15-17 2010

Are Malware a Serious and Real Threat?

Are Malware a Serious and Real Threat?

- In the early days malware were mostly created as pranks or vandalism attempts
  - Or to brag ourselves :-)
- AV companies usually won by developing syntactic signatures

Are Malware a Serious and Real Threat?

Image Copyright: IKARUS Security Software GmbH

- Unfortunately, things changed rapidly!
- Clear shift towards profit-driven goals

> *"[. . .] the release rate of malicious code and other unwanted programs may be exceeding that of legitimate software applications", Symantec 2008*

Are Malware a Serious and Real Threat?

KlikTeamParty – 2008

Wait, we know how to defend ourselves. . .

- The AV industry is moving towards behavioral solutions
- Unfortunately, malware can still slip under the radar (perfect detectors do not exist)
  - New evasion techniques

Wait, we know how to defend ourselves. . .

- The AV industry is moving towards behavioral solutions
- Unfortunately, malware can still slip under the radar (perfect detectors do not exist)
  - New evasion techniques
- Moreover, what to do if we **suspect** a system is compromised?

Wait, we know how to defend ourselves. . .

- The AV industry is moving towards behavioral solutions
- Unfortunately, malware can still slip under the radar (perfect detectors do not exist)
  - New evasion techniques
- Moreover, what to do if we **suspect** a system is compromised?
  - Forensic analysis
  - We all operate at the same privilege level. . .

. . . it is like a dog chasing its tail!

Wait, we know how to defend ourselves. . .

- The AV industry is moving towards behavioral solutions
- Unfortunately, malware can still slip under the radar (perfect detectors do not exist)
  - New evasion techniques
- Moreover, what to do if we **suspect** a system is compromised?
  - Forensic analysis
  - We all operate at the same privilege level. . .

. . . it is like a dog chasing its tail!

We must operate at a privilege level **higher** than the malware

Virtualization comes (again, back) to help

- To analyze malicious samples and provide valuable information
  (e.g., Anubis, CWSandbox, Wepawet)
- To monitor the guests
  (e.g., ReVirt, Ether)
- To protect the guests from attacks
  (e.g., SecVisor)
- **To run forensics analyses**

Virtualization comes (again, back) to help

- To analyze malicious samples and provide valuable information (e.g., Anubis, CWSandbox, Wepawet)
- To monitor the guests (e.g., ReVirt, Ether)
- To protect the guests from attacks (e.g., SecVisor)
- **To run forensics analyses**

Unfortunately. . .

Virtualization comes (again, back) to help

Unfortunately. . .

The target system must be already running inside a VM!

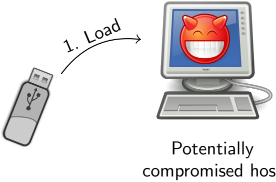
- What can we do?
  - Shut the system off and analyze it off-line
    - What about all the volatile information?
      (e.g., open files, registry keys, network connections, processes)
  - What about production systems that cannot be shut down?
  - What about production systems that cannot be frozen?

Our Contribution: HyperSleuth

A framework to perform live and trustworthy forensic analyses of commodity production systems

Our Contribution: HyperSleuth

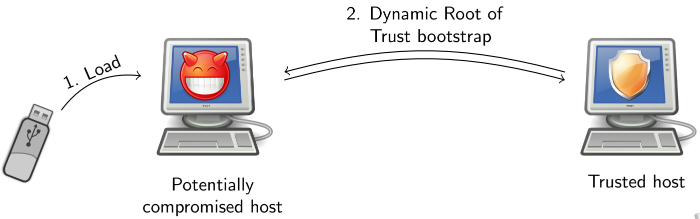
A framework to perform live and trustworthy forensic analyses of commodity production systems



HyperSleuth is installed on an allegedly compromised target as the target system runs

Our Contribution: HyperSleuth

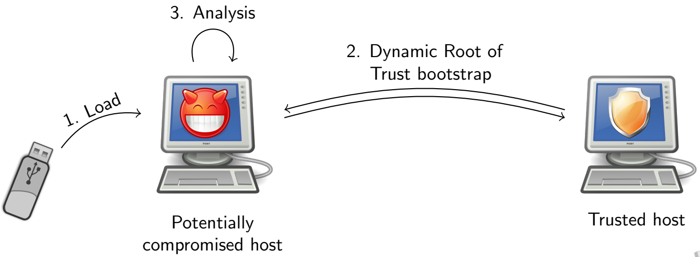
A framework to perform live and trustworthy forensic analyses of commodity production systems


1. Load
2. Dynamic Root of Trust bootstrap
Potentially compromised host
Trusted host

The installation of HyperSleuth is attested with the help of a trusted host

Our Contribution: HyperSleuth

A framework to perform live and trustworthy forensic analyses of commodity production systems



3. Analysis

1. Load

2. Dynamic Root of Trust bootstrap

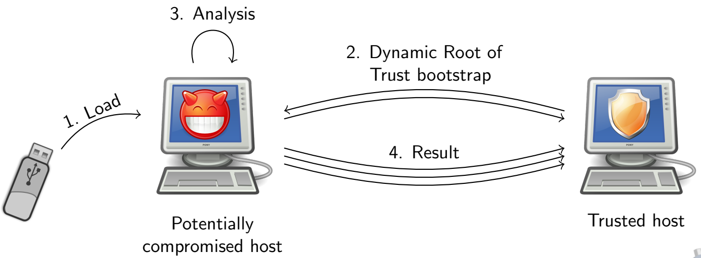
Potentially compromised host

Trusted host

The analyzed OS needs not to be modified at all, and applications continue to run with no service disruption

Our Contribution: HyperSleuth

A framework to perform live and trustworthy forensic analyses of commodity production systems


3. Analysis
1. Load
2. Dynamic Root of Trust bootstrap
4. Result
Potentially compromised host
Trusted host

At the end of the analysis, the results can be sent to the trusted host

How?

Exploit hardware support for virtualization

1. A tiny hypervisor
2. A secure loader that installs the hypervisor
   - It verifies the hypervisor's code, data and its environment

- The forensic framework runs at the hypervisor privilege level (it is more privileged than the OS and completely isolated)
  - Lazy physical memory dumper
  - Lie detector
  - System call tracer (not discussed in this talk)

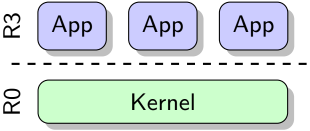
How?

Exploit hardware support for virtualization

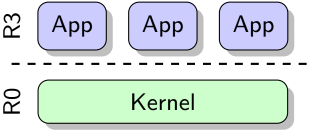
1. A tiny hypervisor
2. A secure loader that installs the hypervisor
    - It verifies the hypervisor's code, data and its environment

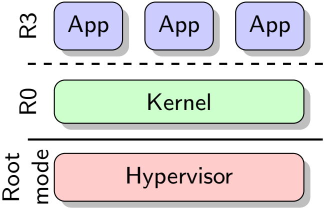
- The forensic framework runs at the hypervisor privilege level (it is more privileged than the OS and completely isolated)
    - Lazy physical memory dumper
    - Lie detector
    - System call tracer (not discussed in this talk)

A Glimpse at Hardware-assisted Virtualization (Intel VT-x)

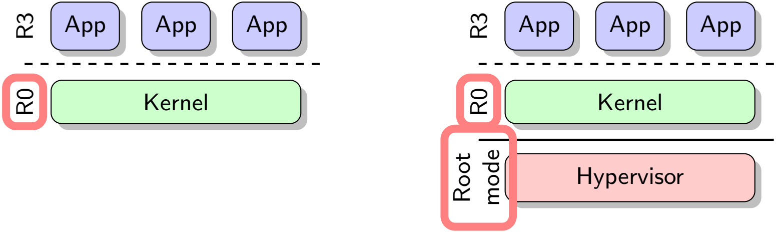


A Glimpse at Hardware-assisted Virtualization (Intel VT-x)



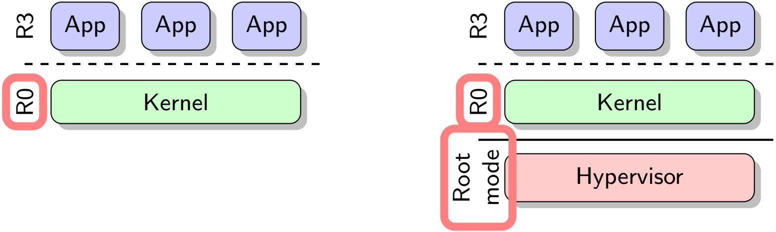


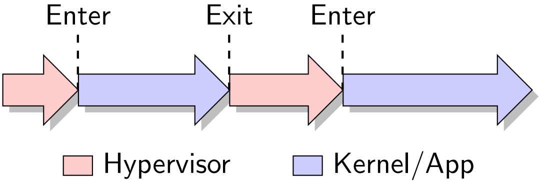
A Glimpse at Hardware-assisted Virtualization (Intel VT-x)



- The OS needs not to be modified
- Minimal overhead
- The hardware guarantees transparency & isolation
- Available on commodity x86 CPUs

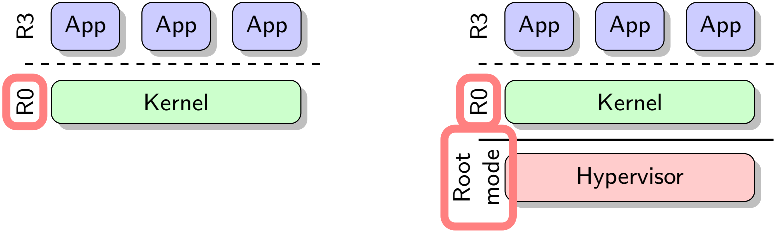
A Glimpse at Hardware-assisted Virtualization (Intel VT-x)

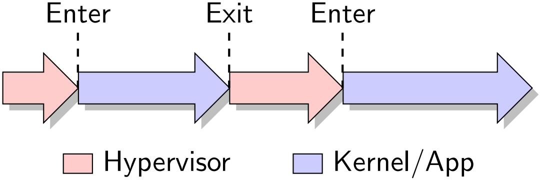




An exit/enter event causes the CPU to save the state of the guest/host inside the VMCS

A Glimpse at Hardware-assisted Virtualization (Intel VT-x)





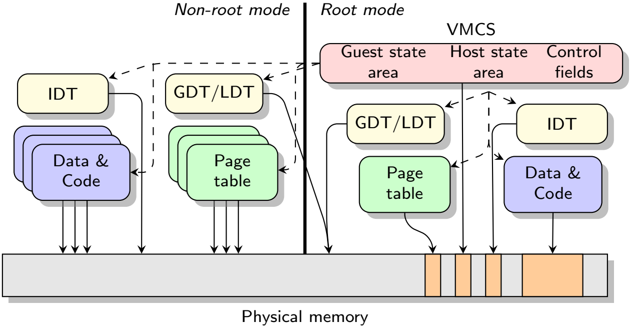
The events that trigger an exit to root mode can be configured dynamically

HyperSleuth Virtual Machine Monitor

- Software-based MMU virtualization through shadow PTs
- Unrestricted guest access to I/O devices
- Direct network access
- VMM on-the-fly removal

HyperSleuth Virtual Machine Monitor

- Software-based MMU virtualization through shadow PTs
- Unrestricted guest access to I/O devices
- Direct network access
- VMM on-the-fly removal



VMM code/data **isolation** from the guest OS

(i.e., VMM can access guest's resources, but not the other way around)

How?

Exploit hardware support for virtualization

1. A tiny hypervisor
2. A secure loader that installs the hypervisor
   - It verifies the hypervisor's code, data and its environment

- The forensic framework runs at the hypervisor privilege level (it is more privileged than the OS and completely isolated)
  - Lazy physical memory dumper
  - Lie detector
  - System call tracer (not discussed in this talk)

Trusted Execution Environment

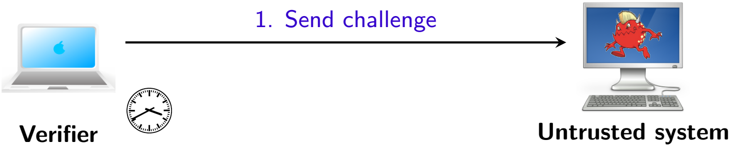
The loader provides a trusted execution environment (TEE)

- Provides a Dynamic Root of Trust (DRT) for live analyses

Characteristics

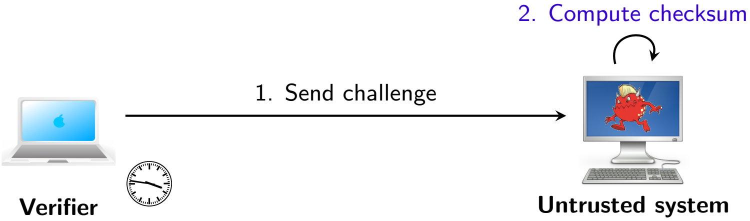
1. Tamper-proof execution of HyperSleuth and its analyses
2. Aposteriori bootstrap of the TEE, aka *late launch*
3. Transparency to the system and attacker
4. Persistency

Software-based Attestation through Challenge-Response



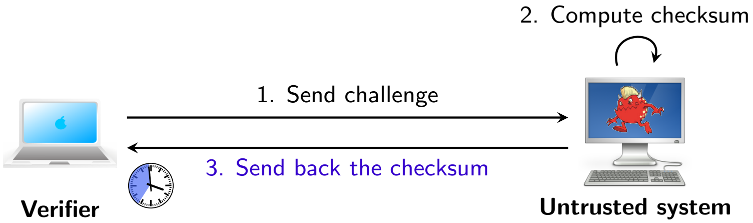
- The verifier challenges the untrusted system (to compute a checksum)

Software-based Attestation through Challenge-Response



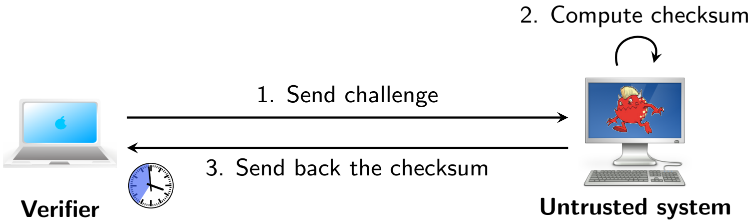
- The untrusted system executes the *checksum function*
- Should be executed at the **highest level of privilege**
- Should execute **without any interruption**

Software-based Attestation through Challenge-Response



- The checksum must be received within a **time interval**
- Time is measured by an external entity (the *verifier*)
- If the checksum is wrong or the timeout has expired, attestation fails

Software-based Attestation through Challenge-Response



- The checksum must be received within a **time interval**
- Time is measured by an external entity (the *verifier*)
- If the checksum is wrong or the timeout has expired, attestation fails

Any attempt to tamper the execution environment results in a noticeable overhead in checksum computation

Live Forensic Analyses

Exploit hardware support for virtualization

1. A tiny hypervisor
2. A secure loader that installs the hypervisor
   - It verifies the hypervisor's code, data and its environment

- The forensic framework runs at the hypervisor privilege level (it is more privileged than the OS and completely isolated)
  - Lazy physical memory dumper
  - Lie detector
  - System call tracer (not discussed in this talk)

Physical Memory Dumper

- Traditional approaches for dumping physical memory have drawbacks
  - PCI cards
  - FireWire devices
  - Kernel drivers
- Tricky problem: memory dumps should be done atomically
  - To guarantee the integrity of the dumped data
  - To avoid attacker's interference with the analysis and results

Physical Memory Dumper

- Traditional approaches for dumping physical memory have drawbacks
  - PCI cards
  - FireWire devices
  - Kernel drivers
- Tricky problem: memory dumps should be done atomically
  - To guarantee the integrity of the dumped data
  - To avoid attacker's interference with the analysis and results
- Atomic memory dumps are likely to freeze the system
  - Time-consuming, esp. when marginal evidence of compromise
  - Consequent money loss and dangerous

HyperSleuth's Lazy Physical Memory Dumper

- Lazily dumps the content of physical memory
  - The CPU is not monopolized
  - Processes running in the system are not interrupted

State of dumped physical memory ≡ state of physical memory
at the time the dump is requested

HyperSleuth's Lazy Physical Memory Dumper

- Lazily dumps the content of physical memory
  - The CPU is not monopolized
  - Processes running in the system are not interrupted

State of dumped physical memory $\equiv$ state of physical memory
at the time the dump is requested

- No process can clean the memory after HyperSleuth is installed (we trap to the hypervisor)
- Memory dumps lazily transmitted via network
  - Compatible with off-the-shelf tools for memory forensic analysis (e.g., Volatility)

HyperSleuth's Lazy Physical Memory Dumper

The algorithm

The algorithm is loosely inspired by the OS' Copy-on-Write

- Dump-on-Write (DOW)
  (i.e., dump the page before it is modified by the guest)
- Dump-on-Idle (DOI)
  (i.e., dump the page when the guest is idle)

HyperSleuth's Lazy Physical Memory Dumper

The algorithm

```
switch (VMM exit reason)
  case CR3 write:
     Sync PT and SPT
     for (v = 0; v < sizeof(SPT); v++)
        if (SPT[v].Writable && !DUMPED[SPT[v].PhysicalAddress])
           SPT[v].Writable = 0;
  case Page fault: // 'v' is the faulty address
     if (PT/SPT access)
        Sync PT and SPT and protect SPTEs if necessary
     else if (write access && PT[v].Writable)
        if (!DUMPED[PT[v].PhysicalAddress])
           DUMP(PT[v].PhysicalAddress);
        SPT[v].Writable = DUMPED[PT[v].PhysicalAddress] = 1;
     else
        Pass the exception to the OS
  case Hlt:
     for (p = 0; p < sizeof(DUMPED); p++)
        if (!DUMPED[p])
           DUMP(p); DUMPED[p] = 1;
           break;
```

The VMM intercepts updates of the page table address, page-fault exceptions, and CPU idle loops

HyperSleuth's Lazy Physical Memory Dumper

The algorithm

```
switch (VMM exit reason)
  case CR3 write:
     Sync PT and SPT
     for (v = 0; v < sizeof(SPT); v++)
        if (SPT[v].Writable && !DUMPED[SPT[v].PhysicalAddress])
           SPT[v].Writable = 0;
  case Page fault: // 'v' is the faulty address
     if (PT/SPT access)
        Sync PT and SPT and protect SPTEs if necessary
     else if (write access && PT[v].Writable)
        if (!DUMPED[PT[v].PhysicalAddress])
           DUMP(PT[v].PhysicalAddress);
        SPT[v].Writable = DUMPED[PT[v].PhysicalAddress] = 1;
     else
        Pass the exception to the OS
  case Hlt:
     for (p = 0; p < sizeof(DUMPED); p++)
        if (!DUMPED[p])
           DUMP(p); DUMPED[p] = 1;
           break;
```

During a context switch (CR3 update) the algorithm grants **read-only** permissions to physical not yet dumped pages

HyperSleuth's Lazy Physical Memory Dumper

The algorithm

```
switch (VMM exit reason)
  case CR3 write:
     Sync PT and SPT
     for (v = 0; v < sizeof(SPT); v++)
        if (SPT[v].Writable && !DUMPED[SPT[v].PhysicalAddress])
           SPT[v].Writable = 0;
  case Page fault: // 'v' is the faulty address
     if (PT/SPT access)
        Sync PT and SPT and protect SPTEs if necessary
     else if (write access && PT[v].Writable)
        if (!DUMPED[PT[v].PhysicalAddress])
           DUMP(PT[v].PhysicalAddress);
        SPT[v].Writable = DUMPED[PT[v].PhysicalAddress] = 1;
     else
        Pass the exception to the OS
  case Hlt:
     for (p = 0; p < sizeof(DUMPED); p++)
        if (!DUMPED[p])
           DUMP(p); DUMPED[p] = 1;
           break;
```

Our write protection is reinforced after every update of the page tables

HyperSleuth's Lazy Physical Memory Dumper

The algorithm

```
switch (VMM exit reason)
  case CR3 write:
     Sync PT and SPT
     for (v = 0; v < sizeof(SPT); v++)
        if (SPT[v].Writable && !DUMPED[SPT[v].PhysicalAddress])
           SPT[v].Writable = 0;
  case Page fault: // 'v' is the faulty address
     if (PT/SPT access)
        Sync PT and SPT and protect SPTEs if necessary
     else if (write access && PT[v].Writable)
        if (!DUMPED[PT[v].PhysicalAddress])
           DUMP(PT[v].PhysicalAddress);
        SPT[v].Writable = DUMPED[PT[v].PhysicalAddress] = 1;
     else
        Pass the exception to the OS
  case Hlt:
     for (p = 0; p < sizeof(DUMPED); p++)
        if (!DUMPED[p])
           DUMP(p); DUMPED[p] = 1;
           break;
```

Write accesses to pages not yet dumped trigger **page fault** exceptions, and pages are dumped before being modified (DOW)

HyperSleuth's Lazy Physical Memory Dumper

The algorithm

```
switch (VMM exit reason)
  case CR3 write:
     Sync PT and SPT
     for (v = 0; v < sizeof(SPT); v++)
        if (SPT[v].Writable && !DUMPED[SPT[v].PhysicalAddress])
           SPT[v].Writable = 0;
  case Page fault: // 'v' is the faulty address
     if (PT/SPT access)
        Sync PT and SPT and protect SPTEs if necessary
     else if (write access && PT[v].Writable)
        if (!DUMPED[PT[v].PhysicalAddress])
           DUMP(PT[v].PhysicalAddress);
        SPT[v].Writable = DUMPED[PT[v].PhysicalAddress] = 1;
     else
        Pass the exception to the OS
  case Hlt:
     for (p = 0; p < sizeof(DUMPED); p++)
        if (!DUMPED[p])
           DUMP(p); DUMPED[p] = 1;
           break;
```

To guarantee termination, pending pages are dumped on CPU idle loops

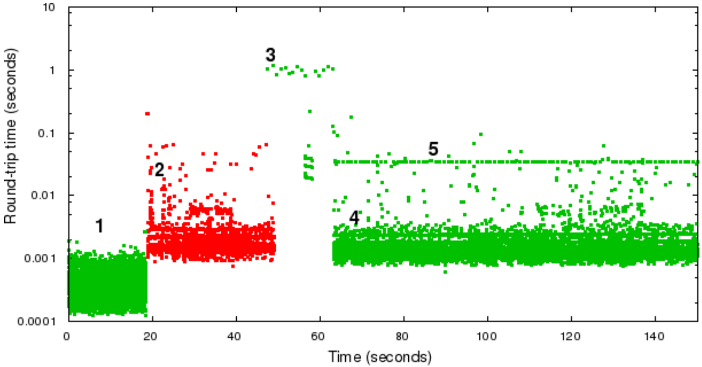
HyperSleuth's Lazy Physical Memory Dumper

Experimental setup

- Current implementation of HyperSleuth specific to Microsoft Windows XP (32-bit)
- Hardware features of the host running HyperSleuth
  - Intel CPU Core i7
  - 3GB Ram
  - Realtek RTL8139 100Mbps network card
- Trusted host is a common laptop machine
- DNS server was compromised and subjected to the heavy loads

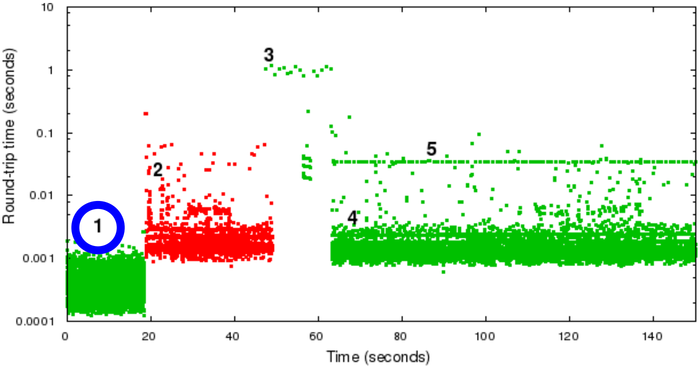
HyperSleuth's Lazy Physical Memory Dumper

Evaluation



HyperSleuth's Lazy Physical Memory Dumper

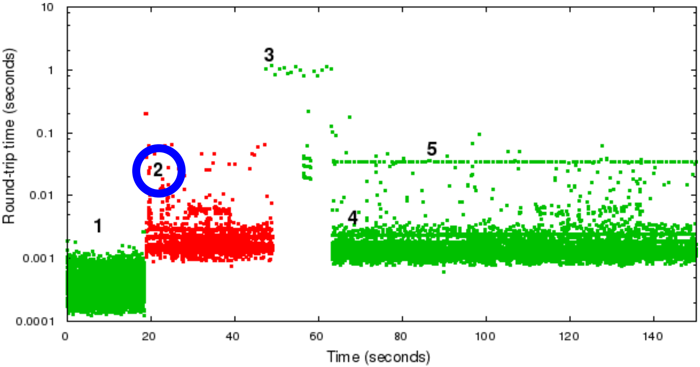
Evaluation



Before launching HyperSleuth, the average round-trip time was $\sim 0.34ms$

HyperSleuth's Lazy Physical Memory Dumper

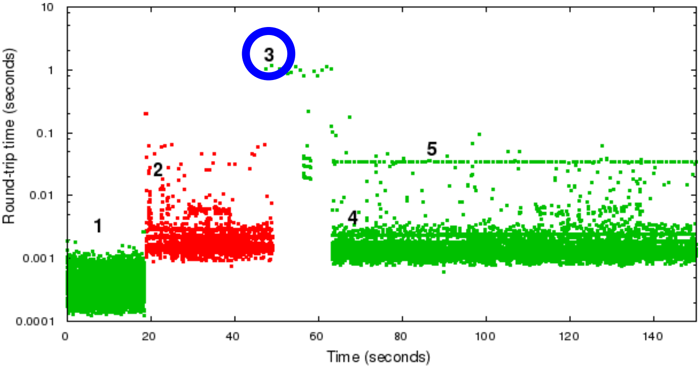
Evaluation



DRT bootstrap and the installation of the VMM ($\sim 0.19s$),
then RTT stabilized around $1.6ms$

HyperSleuth's Lazy Physical Memory Dumper

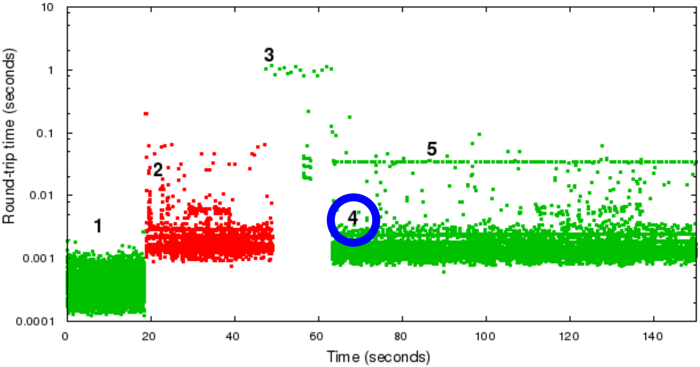
Evaluation



When we started the dump, a lot of frequently accessed pages were dumped

HyperSleuth's Lazy Physical Memory Dumper

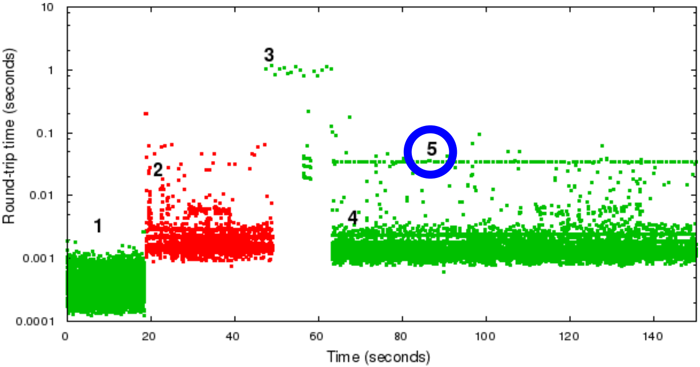
Evaluation



Then, RTT stabilized again around 1.6*ms*

HyperSleuth's Lazy Physical Memory Dumper

Evaluation



Regular peaks ($\sim 32ms$) were caused by periodic dump of non-written pages

HyperSleuth's Lazy Physical Memory Dumper

Evaluation

- The system never entered the idle loop (heavy load)
  - Configured to dump at least 64 pages every second
- Whole physical memory dump in about 180 minutes

HyperSleuth's Lazy Physical Memory Dumper

Evaluation

- The system never entered the idle loop (heavy load)
  - Configured to dump at least 64 pages every second
- Whole physical memory dump in about 180 minutes
- Non-negligible overhead, but **no** service interruption
  - No DNS request-reply timed out
  - Decreasing dumping time possible with higher RTT
  - Possibly 640 pages/sec on a 1Gbps media with no add. overhead
    - 3GB RAM dumped in about 18mins with **no** service interruption

HyperSleuth's Lazy Physical Memory Dumper

Evaluation

- The system never entered the idle loop (heavy load)
  - Configured to dump at least 64 pages every second
- Whole physical memory dump in about 180 minutes
- Non-negligible overhead, but **no** service interruption
  - No DNS request-reply timed out
  - Decreasing dumping time possible with higher RTT
  - Possibly 640 pages/sec on a 1Gbps media with no add. overhead
    - 3GB RAM dumped in about 18mins with **no** service interruption
- Traditional, atomic, dumping approaches would have taken
  - 24s, 50s, 4mins on a 1Gbps, 480Mbps, 100Mbps, respectively
  - No real guarantee on the integrity of the dump...

Lies, lies, nothing but lies!

- Kernel-level malware insidious and dangerous
  - Operate at a very high privilege level
  - Able to hide any resource an attacker wants to protect (e.g., processes, network communications, files)
- Different techniques to force the OS to lie about its state
- How can we disguise such liars?
  - Retrieve $\mathcal{S}_{guest}$, the state perceived by the (guest) system
  - Retrieve $\mathcal{S}_{VMM}$, the state perceived by the VMM
  - $\mathcal{S}_{guest} = \mathcal{S}_{VMM}$?

HyperSleuth's Lie Detector

- HyperSleuth's loader runs a minimalistic in-guest utility
  - Collects the state of the system as perceived by the guest
  - Such information is sent to the trusted host
  - The utility makes an hypercall that causes a VM exits
- HyperSleuth's loader establishes the TEE and launch the VMM
  - System's state is collected from within the VMM (OS-aware inspection)
  - Results are sent back to the trusted host
- Diffs ? "infected" : "not infected"

HyperSleuth's Lie detector

Evaluation

Sample	Characteristics	Detected?
FU	DKOM	✓
FUTo	DKOM	✓
HaxDoor	DKOM, SSDT hooking, API hooking	✓
HE4Hook	SSDT hooking	✓
NtIllusion	DLL injection	✓
NucleRoot	API hooking	✓
Sinowal	MBR infection, Run-time patching	✓

HyperSleuth's Lie detector

Evaluation

Sample	Characteristics	Detected?
FU	DKOM	✓
FUTo	DKOM	✓
HaxDoor	DKOM, SSDT hooking, API hooking	✓
HE4Hook	SSDT hooking	✓
NtIllusion	DLL injection	✓
NucleRoot	API hooking	✓
Sinowal	MBR infection, Run-time patching	✓

FUTo leverages DKOM to hide malicious resources. We scan Windows' internal structures that must be left intact to preserve system functionalities

HyperSleuth's Lie detector

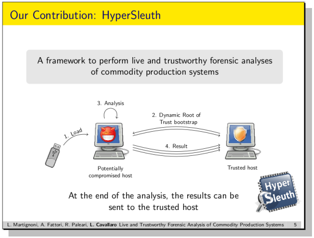
Evaluation

Sample	Characteristics	Detected?
FU	DKOM	✓
FUTo	DKOM	✓
HaxDoor	DKOM, SSDT hooking, API hooking	✓
HE4Hook	SSDT hooking	✓
NtIllusion	DLL injection	✓
NucleRoot	API hooking	✓
Sinowal	MBR infection, Run-time patching	✓

HaxDoor hooks system calls and filters their result. We observed hidden registry keys were missing from the untrusted view.

Conclusions

Conclusions



Our Contribution: HyperSleuth

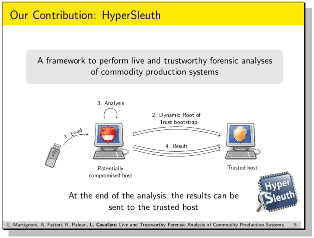
A framework to perform live and trustworthy forensic analyses of commodity production systems

3. Analysis
1. Load
2. Dynamic Root of Trust bootstrap
4. Result
Potentially compromised host
Trusted host
Hyper Sleuth

At the end of the analysis, the results can be sent to the trusted host

L. Martignoni, A. Fattori, R. Paleari, **L. Cavallaro** Live and Trustworthy Forensic Analysis of Commodity Production Systems 5

Conclusions

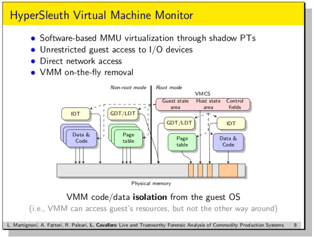


Our Contribution: HyperSleuth

A framework to perform live and trustworthy forensic analyses of commodity production systems

3. Analysis
1. Load
2. Dynamic Root of Trust bootstrap
4. Result
Potentially compromised host
Trusted host
Hyper Sleuth

At the end of the analysis, the results can be sent to the trusted host



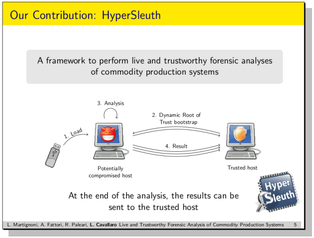
HyperSleuth Virtual Machine Monitor

- Software-based MMU virtualization through shadow PTs
- Unrestricted guest access to I/O devices
- Direct network access
- VMM on-the-fly removal

VMM code/data **isolation** from the guest OS

(i.e., VMM can access guest's resources, but not the other way around)

Conclusions

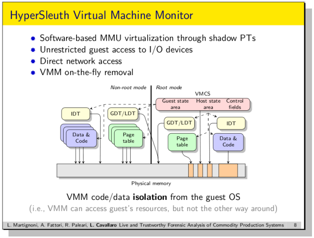

3. Analysis
1. Load
2. Dynamic Root of Trust bootstrap
4. Result
Potentially compromised host
Trusted host
Hyper Sleuth

Our Contribution: HyperSleuth

A framework to perform live and trustworthy forensic analyses of commodity production systems

At the end of the analysis, the results can be sent to the trusted host

L. Martignoni, A. Fattori, R. Paleari, **L. Cavallaro** Live and Trustworthy Forensic Analysis of Commodity Production Systems 5



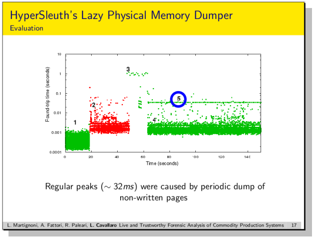
HyperSleuth Virtual Machine Monitor

- Software-based MMU virtualization through shadow PTs
- Unrestricted guest access to I/O devices
- Direct network access
- VMM on-the-fly removal

VMM code/data **isolation** from the guest OS

(i.e., VMM can access guest's resources, but not the other way around)

L. Martignoni, A. Fattori, R. Paleari, **L. Cavallaro** Live and Trustworthy Forensic Analysis of Commodity Production Systems 8



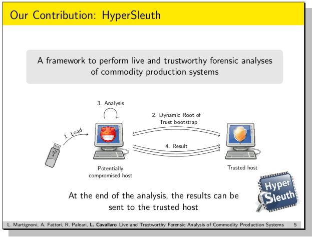
HyperSleuth's Lazy Physical Memory Dumper

Evaluation

Regular peaks (∼ 32*ms*) were caused by periodic dump of non-written pages

L. Martignoni, A. Fattori, R. Paleari, **L. Cavallaro** Live and Trustworthy Forensic Analysis of Commodity Production Systems 17

Conclusions

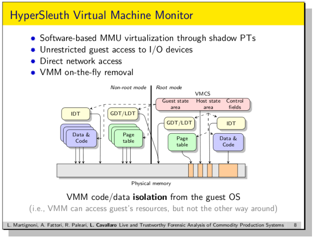


Our Contribution: HyperSleuth

A framework to perform live and trustworthy forensic analyses of commodity production systems

3. Analysis
1. Load
2. Dynamic Root of Trust bootstrap
4. Result
Potentially compromised host
Trusted host
Hyper Sleuth

At the end of the analysis, the results can be sent to the trusted host

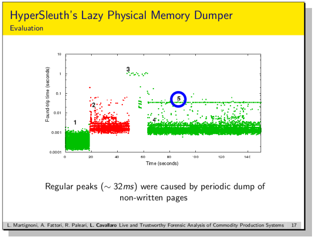


HyperSleuth Virtual Machine Monitor

- Software-based MMU virtualization through shadow PTs
- Unrestricted guest access to I/O devices
- Direct network access
- VMM on-the-fly removal

VMM code/data **isolation** from the guest OS

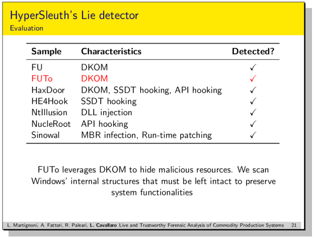
(i.e., VMM can access guest's resources, but not the other way around)



HyperSleuth's Lazy Physical Memory Dumper

Evaluation

Regular peaks (~ 32*ms*) were caused by periodic dump of non-written pages



HyperSleuth's Lie detector

Evaluation

Sample	Characteristics	Detected?
FU	DKOM	✓
FUTo	DKOM	✓
HaxDoor	DKOM, SSDT hooking, API hooking	✓
HE4Hook	SSDT hooking	✓
NtIllusion	DLL injection	✓
NucleRoot	API hooking	✓
Sinowal	MBR infection, Run-time patching	✓

FUTo leverages DKOM to hide malicious resources. We scan Windows' internal structures that must be left intact to preserve system functionalities

Live and Trustworthy Forensic Analysis of Commodity Production Systems

Thank you!
Any questions?

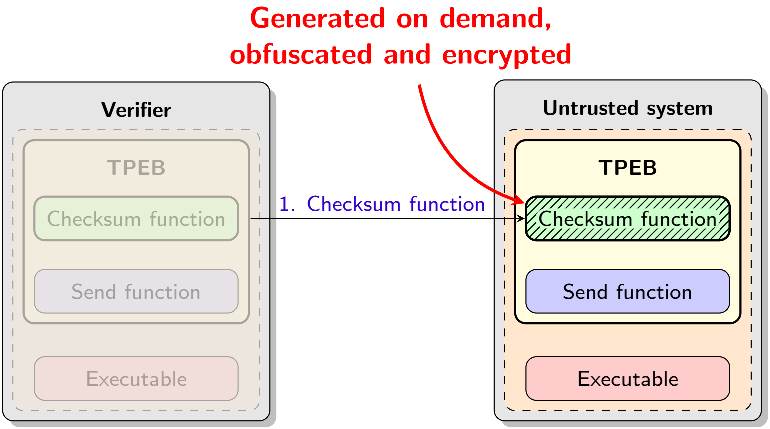
Lorenzo Cavallaro
<sullivan@cs.vu.nl>

Backup slides

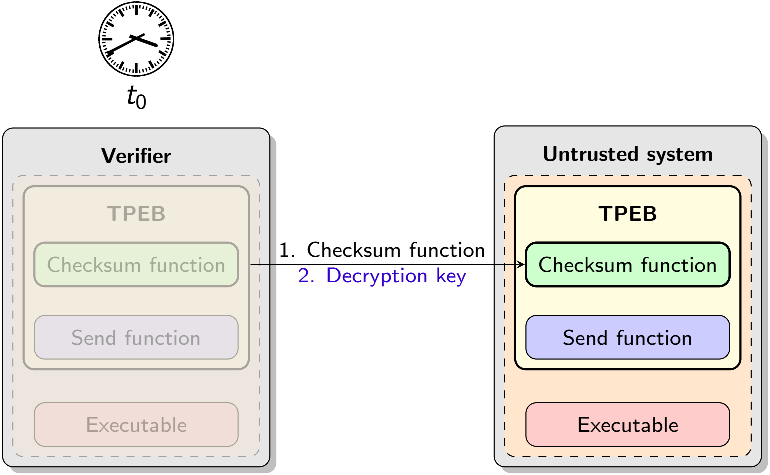
How Does Conqueror Work?

- Variation of the traditional challenge-response scheme
- The challenge is not a seed, but consists in the **whole** checksum function
- The checksum function is:
    1. Generated on demand
    2. Obfuscated
    3. Self-decrypting

Conqueror Protocol



Conqueror Protocol



t_0

Verifier

TPEB

Checksum function

Send function

Executable

1. Checksum function

2. Decryption key

Untrusted system

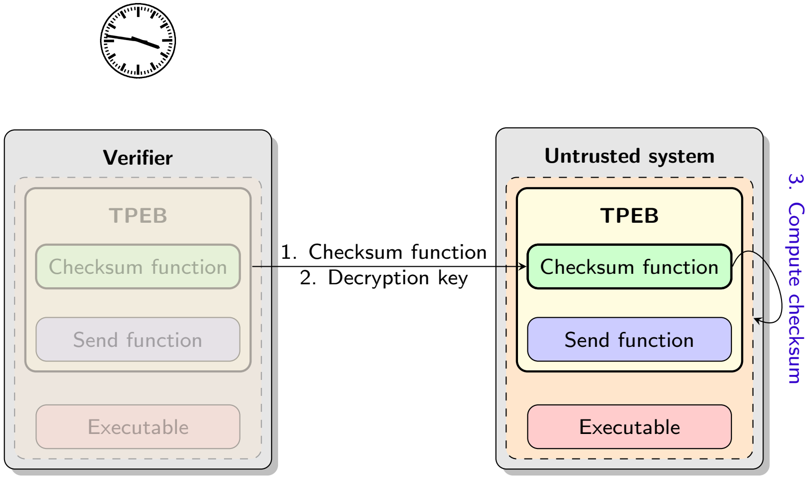
TPEB

Checksum function

Send function

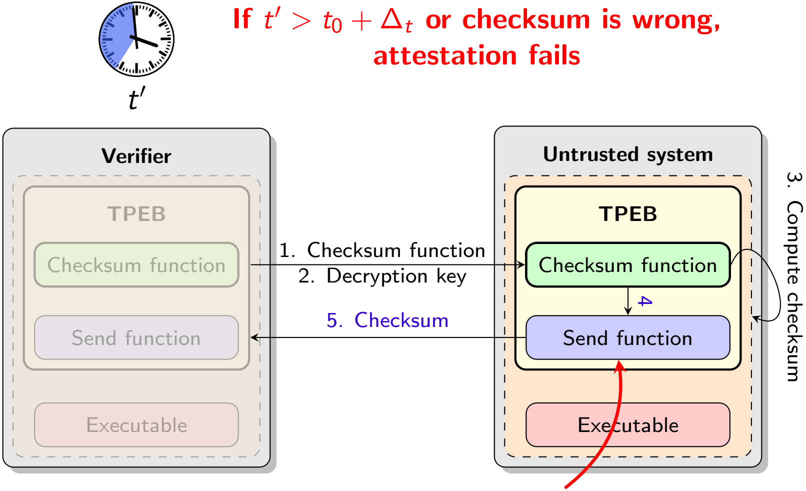
Executable

Conqueror Protocol



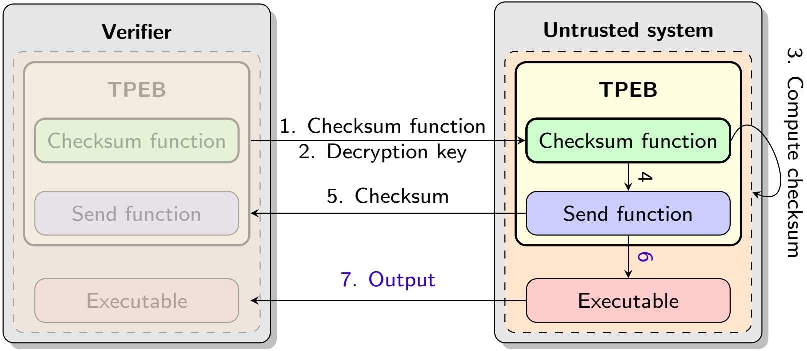
Conqueror Protocol

If $t' > t_0 + \Delta_t$ or checksum is wrong, attestation fails



Hardware-dependent

Conqueror Protocol


Verifier
TPEB
Checksum function
Send function
Executable
1. Checksum function
2. Decryption key
5. Checksum
7. Output
Untrusted system
TPEB
Checksum function
4
Send function
6
Executable
3. Compute checksum