TF-BIV: Transparent and Fine-grained Binary Integrity Verification in the Cloud

Fangjie Jiang, Quanwei Cai, Jingqiang Lin, Bo Luo, Le Guan, Ziqiang Ma
Motivation

- Cloud computing service
  - 94% (IT Inc.)
  - 1.2 billion (2013-2018)

- Add-on services
  - Mail server
  - Developer tools
  - Database
  - Internet of things
Motivation

- **Cryptographic** Cloud Services – **Sensitive** Service
  - AWS CloudHSM, Alibaba Aliyun encryption service
  - Key Management Service (KMS)
    - AWS KMS, Alibaba Aliyun KMS, Microsoft Azure Key Vault, etc.
  - key security
    - Strong enough
    - Cryptographic algorithms
    - FIPS 140-2 HSM
- **Invocation security**
  - ID and PW.
- **Binary Integrity as a Service**
  - Cloud service provider: protecting platform security
  - Tenants: protecting critical service
  - **Authorization** of callers of sensitive services: specified and enforced at the **process level**
Challenges

- Desired properties
  - **Isolation**
    - Isolated from guest VM
  - **Transparency**
    - No modification
    - Guest OS
    - Target application
  - **TOCTTOU consistency**
    - Time of Check to Time of Use
    - Beyond the time-of-verification
    - Entire lifetime
  - **Fine-grained**
    - Designate the sensitive applications (called S-process) to be protected

<table>
<thead>
<tr>
<th></th>
<th>Isolation</th>
<th>Consistency</th>
<th>Transparency</th>
<th>Fine-grained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patagonix (08’Usenix Securit)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>HIMA (09’ACSAC)</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>En-ACCI (18’ISC)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>InkTag (13’ASPLOS)</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>AppShield (15’AsiaCCS)</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>AppSec (15’VEE)</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>

○: not supported ●: partially supported ●: fully supported
BACKGROUND

- Intel Hardware-assisted Virtualization
  - Virtual machine control structure (VMCS)
    - Monitor trap flag (MTF)
      - Single step
    - CR3-load exiting flag
      - MOV to CR3
  - Extended Page Table (EPT)
    - Guest VM
      - Guest virtual address(GVA) → Guest physical address (GPA)
    - EPT
      - GPA → Host physical address(HPA)
    - Privilege flags
      - Read, write, and execute
      - EPT violation
TF-BIV

- **Overview**
  - Based on hardware event
    - Performed in VMM (Isolation)
    - No modifications in VM (Transparency)
  - Intel EPT W RX
    - Verification before execution
    - Capturing any modifications to verified code (TOCTTOU consistency)
  - Comprehensive Verification of S-process (Fine-grained)
    - S-process creation, switch identification
      - CR3-load exiting
    - Verification of related codes
      - Loadable kernel module, kernel, shared library, mixed page
      - Monitoring any update to page table
        - Intel EPT, MTF
TF-BIV

- **Initialization**
  - Generating reference hash values

- **Running time**
  1. **S-process creation**
     - CR3-load exit event
  2. **S-process’s page table updating**
     - EPT non-writable exit event
     - MTF exit event
  3. **Code execution in S-process’s address space**
     - EPT non-executable exit event
  4. **TF-BIV capturing all modifications to the verified code**
     - EPT non-writable exit event
TF-BIV

- S-process identification
  - CR3-load exiting
    - Unknown CR3 value
      - Creation of a new process
    - S-process identification
      - VMI
      - Directly but inaccurate
      - Reference hash
TF-BIV

- Memory layout monitoring
  - Monitoring page table update
    - EPT non-writable exiting + MTF exiting
    - Paging structures setting non-writable
      - Capturing guest VM updating
    - EPT non-writable exiting
      - Allowing write access
      - Enable MTF exiting
    - MTF exiting
      - Checking updating status
      - Disable MTF
        - Paging structures again non-writable
        - Monitor further page table update
    - Timely discover new mapped code page
Identifying newly mapped memory areas

- New mapped code page identification
  - Find the corresponding reference hash

- Native
  - One by one
  - Time-consuming

- Optimization based on VMI
  - Obtain the corresponding binary information
    - Binary name, start VA .etc.
  - Acceleration
  - No influence of the verification result
TF-BIV

- Integrity verification of code pages
  - WØX
  - New loaded guest physical page – non-executable
  - EPT non-executable exiting
    - Virtual address
      - User space
        - S-process’s related code
        - S-process
        - Shared libraries
      - Nonrelated code
    - Kernel space

Flowchart:

1. Page content, GVA, GPA, CR3 value
2. Kernel code
   - Yes → Kernel code verification
   - No → CR3 value in S-process list
     - Yes → Integrity verification
     - No → S-process related code
       - Yes → Exit checking
       - No → Memory areas identified
         - Yes → Integrity verification
         - No → Remove from S-process list
**TF-BIV**

- **Integrity Verification of Code Pages**
  - Shared code pages verification
    - Executable before mapped
      - After mapped by S-process
      - Non-executable again
    - Pre-executed by another process
      - Memory layout
        - Binary identifier
        - Offset in the binary
        - GPA of the shared code pages
        - Status (verified/unverified)
        - Count of S-process
          - Mapped the shared code pages
          - Once count is 0, remove the record

**Diagram:**

```
Page content, GVA, GPA, CR3 value

Kernel code
  +-----------------+    Yes
  | Kernel code verification |
  +-----------------+    No

CR3 value in S-process list
  - No

S-process related code
  +-----------------+    Yes
  | S-process related code verification |
  +-----------------+    No

Memory areas identified
  - No

Integrity verification
  +-----------------+    Yes
  | Exit checking |
  +-----------------+    No

Remove from S-process list
```
TF-BIV

Kernel Space integrity
  - Kernel
    - VM startup
    - Non-writable
  - Loadable kernel module (LKM)
    - Dynamically loaded kernel component
    - Relocation and linking
      - Init block (erased after LKM loaded)
      - Code block
    - Recording all loaded LKM information
      - Write to the memory area of a verified LKM
        - Remove its information
        - Unloaded
TF-BIV

- Mixed page
  - Code and mutable data: W ⊕ X
  - Solution
    - Original mixed page provides data view (NX)
    - New physical page provides code view (NW)
      - Duplication from original mixed page
      - Replace data region with NOP
    - View switch
      - EPT non-executable exiting
        - Code integrity verification
        - Code view
        - EPT non-writable + non-readable exiting
          - Data view
      - iTLB、dTLB
Security Analysis

- All the related code
  - Kernel and LKM
  - Track S-process’s creation
  - Monitor all the page table updates
  - Write protection of verified pages
  - Focus on static code

- Attacks
  - Before loaded into memory
    - Modify the binary of the program during software downloading
  - Runtime
    - Inject malicious code into S-process’s address space
      - Buffer overflow, format string overflow
    - Address mapping manipulation attacks
      - Mapping unverified code page
      - Double mapping or reorder
Application in the Cloud Cryptographic Service

- Integration
  - Specify S-processes
  - TF-BIV provides Integrity verification
  - HTTPS
    - Access control performed in network card emulation
    - Challenge: Couple TF-BIV and Cryptographic service invocation
      - Couple network connections with S-process
      - Further assumption
        - Kernel structures (task_struct, mm_struct, files_struct, fdatable and file)
        - System symbols (init_task, socket_operation, socket_dentry_operation)
        - Kernel data integrity protected by existed schemes Osck or KI-Mon
Implementation

- TF-BIV
  - KVM (Linux 3.13) (1000 lines)
    - `vm_read/vm_write` raed/set VMCS
    - Config EPT access bits
    - Distinguish exit events based on `exit_qlification`

- Integration with cryptographic service
  - e1000 network card emulated in QEMU(1.7.1) (600 lines)
    - Couple VMI and Invocation event
    - Twice check

- Hash Generator
  - Offline
  - ELF
Evaluation

- **Configurations**
  - Dell OptiPlex 9020, Intel i7-4770 (3.4TGHZ), 16GB Memory
  - Host: CentOS 6.6, Linux kernel 3.13, QEMU/KVM 1.7.1
  - VM: CentOS 6.6, Linux kernel 3.13.7, 4vCPU, 4GB Memory

- **Startup**
  - Boot time of VM
  - Bootchart
    - Overhead 1.49%

- **SPECINT Benchmark**
  - Impact about performance of vcpu
  - Comparison
    - Native Linux
    - TF-BIV enabled
    - TF-BIV enable and cryptographic service invoked
  - Overhead 3.6%

![Normalized Overhead](image)
Evaluation -- Network Performance

- Assistant Machine (Intel i5-4590 3.3GHZ, 16GB)
- iPerf
  - Dual Testing and Tradeoff Testing
  - Bandwidth overhead 3.81%

<table>
<thead>
<tr>
<th>Testing Mode</th>
<th>Native</th>
<th>TF-BIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual Testing</td>
<td>C 920.67MB/s</td>
<td>894.00MB/s</td>
</tr>
<tr>
<td></td>
<td>S 632.11MB/s</td>
<td>608.00MB/s</td>
</tr>
<tr>
<td>Tradeoff Testing</td>
<td>C 819.78MB/s</td>
<td>810.56MB/s</td>
</tr>
<tr>
<td></td>
<td>S 1089MB/s</td>
<td>1086MB/s</td>
</tr>
</tbody>
</table>
Evaluation – HTTPS Performance

- Throughout
  - Overhead 8.3% (concurrent 128)

- 99% Latency
  - Overhead 5.7% (concurrent 128)
Conclusion

- The first binary integrity verification scheme achieves the following four desired properties
  - Isolation, transparency, TOUTTOU consistency and fine-grained
- **Authorization** of sensitive services enforced at **process level**
  - Integrated with cloud cryptographic service.
- Prototype system and evaluation
  - 3.81%
Thank You!

Questions?