Experience Report: An Analysis of Hypercall Handler Vulnerabilities

ISSRE 2014, November 4th, Naples, Italy

Aleksandar Milenkoski
Karlsruhe Institute of Technology, Karlsruhe, Germany

Bryan D. Payne
Nebula Inc., Mountain View, CA, USA

Nuno Antunes, Marco Vieira
University of Coimbra, Coimbra, Portugal

Samuel Kounev
University of Würzburg, Würzburg, Germany
Motivation

Virtualization: Benefits and Challenges

- Reduced costs
- Security

VM: Virtual Machine

Attack Scenario: Guest VM → Hypervisor

Hypercall Attacks Are Severe
- Denial-of-service (DoS)
- Malicious code execution

VM exit events / Hypercalls
Motivation

Problem Statement

- Lack of information on hypercall vulnerabilities / attacks
- Characterization of the hypercall attack surface

Contributions

- Analysis of origins of hypercall vulnerabilities
- Demonstration and analysis of hypercall attacks
- Hypercall attack models
- Lessons learned and recommendations
Sample Set of Hypercall Vulnerabilities

Analyzed Hypercall Vulnerabilities
- Xen: 25
- KVM: 9
- Hyper-V: 1

Analysis Approach
(i) Analysis of vulnerability reports
(ii) Reverse-engineering the released patch
(iii) Developing and executing proof-of-concept code

Benefits
- Understanding the origins of hypercall vulnerabilities
- Observation of all stages of an attack life cycle
Hypercall Vulnerabilities: Error Systematization

Systematization of Errors

- Error
  - Implementation 17
  - Non-implementation 18
  - Value validation 16
  - Inverse procedures 1
  - Missing 10
  - Incorrect 6

**Hypercall Vulnerabilities: Example 1**

**Value Validation – Missing**

CVE-2012-5525
- Hypervisor: Xen
- Hypercall: multiple

get_page_from_gfn ← mfn uses mfn as offset

**Discussion**

Variable value validations against missing value validation errors
- Reduced hypercall execution speed
  - Hypercall continuations
- [Xen] Programming practices → Vulnerabilities
- Performance ← Security
Hypercall Vulnerabilities: Example 2

Value Validation – Missing

CVE-2012-5513
Hypervisor: Xen
Hypercall: memory_op

```
copy_to_guest_offset ← addr_to, addr_from
```

addr_from and addr_to pre-validated

```
.HYPERVISOR_update_va_mapping(...)
```

```
HYPERVISOR_memory_op (XENMEM_exchange, &exchange);
```

```
.out.extent_start = 0xFFFF808000000000;
```

```
/*
 * Pre-validate a guest handle.
 * Allows use of faster __copy_* functions.
 */
#define guest_handle_okay(hnd, nr) \  
  (paging_mode_external(current->domain) || \  
   array_access_ok((hnd).p, (nr), sizeof(*((hnd).p))))
```
Hypercall Vulnerabilities: Example 3

Non-implementation

CVE-2013-1964
Hypervisor: Xen
Hypercall: grant_table_op, operation GNTTABOP_copy
copies a page from a source VM (SVM) to a destination VM (DVM) using grants
supports the use of transitive grants that point to a grant of the SVM

What if the acquired grant is non-transitive?

```c
if ( trans_domid == rd->domain_id )
    PIN_FAIL(unlock_out, GNTST_general_error,
        "transitive grants cannot be self-referential\n");

/* We allow the trans_domid == ld->domain_id case, which corresponds to a grant being issued by one domain, sent to another one, and then transitively granted back to the original domain. Allowing it is easy, and means that you don't need to go out of your way to avoid it in the guest. */
```
Non-implementation errors are common

Triggering hypercall vulnerabilities due to non-implementation errors
  “Unexpected” hypercall execution scenarios – regular hypercalls
  Unintentionally during regular system operation
  Are hypercall interfaces reliable?
## Effects of Hypercall Attacks

<table>
<thead>
<tr>
<th>Attack Effect</th>
<th>No. of Hypercall Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash / Corrupted state</td>
<td>12</td>
</tr>
<tr>
<td>Crash</td>
<td>4</td>
</tr>
<tr>
<td>Hang</td>
<td>4</td>
</tr>
<tr>
<td>Corrupted state</td>
<td>4</td>
</tr>
<tr>
<td>Hang / Crash</td>
<td>2</td>
</tr>
<tr>
<td>Crash / Information leakage</td>
<td>1</td>
</tr>
<tr>
<td>Crash / Corrupted state / Information leakage</td>
<td>1</td>
</tr>
</tbody>
</table>

## Discussion

Effective hypervisor DoS attacks

Malicious code execution is probable

A hypercall attacks: A single step of a multi-step attack

Rutkowska and Wojtczuk @ BlackHat 2008

Wilhelm, Luft, and Ray @ HITB 2014
Hypercall Attack Models

Definition and Benefits
Categorized patterns of activities comprising a successful hypercall attack
Facilitate the development of approaches that involve mimicking attackers

The Hypercall Attack Models

Setup phase
One or multiple regular hypercalls
Optional

Attack phase
A single hypercall
with regular parameter values
with specifically crafted parameter value(s)

A series of regular hypercalls in a given order
repetitive execution of a single hypercall
repetitive execution of multiple hypercalls
[Triggering CVE-2012-5513] Setup phase and execution of a single hypercall with a specifically crafted parameter value
[Triggering CVE-2013-1964] Setup phase and execution of a single regular hypercall

**Attack Models: Examples (2/4)**
[Triggering CVE-2013-4494] Execution of a series of regular hypercalls
Attack Models: Examples (4/4)

[Triggering CVE-2012-3495] Repetitive execution of a single regular hypercall
Extending the Frontiers

Securing Hypercall Interfaces: Proactive Approaches
- Preventing hypercall attacks from occurring
  - Vulnerability discovery
  - Secure programming practices

Securing Hypercall Interfaces: Reactive Approaches
- Detecting and preventing hypercall attacks as they occur
  - Hypercall security mechanisms
Securing Hypercall Interfaces: Proactive Approaches

Vulnerability Discovery: Fuzzing

Lack of publicly available approaches and tools for fuzzing hypercalls
Challenge: Systematic coverage of hypercall code

Xen 4.0 release notes

Xen hypervisor 4.0.0 was released on 07 Apr 2010.

Vulnerability Discovery: Formal Verification

Lack of formal verification methods for discovering non-implementation errors

Freitas and McDermott (2011) re-engineer the Xen hypercall interface
Focus: Information-flow security

Pucetti (2010) use Frama-C to analyze Xen hypercalls
Focus: Vulnerabilities due to implementation errors
Securing Hypercall Interfaces: Proactive Approaches

Secure Programming Practices

Programming practices enforcing value validations

- Missing value validation errors: 60% of all implementation errors

Challenge

- Performance (… remember continuations?)

Security enhanced operating mode of hypervisors (XSM FLASK)

- Security-enhanced operating modes of hypercall interfaces?
Securing Hypercall Interfaces: Reactive Approaches

Hypercall Security Mechanisms

Intrusion detection systems and MAC systems
Collabra, MAC/HAT, RandHyp, XSM-FLASK

Attacks triggering vulnerabilities due to non-implementation errors?

Novel hypercall attack detection techniques

Attack models (attack phase)

- Execution of a series of regular hypercall in a given order*
- Repetitive execution of a single hypercall*
- Repetitive execution of multiple hypercalls*
- Execution of a single hypercall with specifically crafted parameter value(s)
- Execution of a single regular hypercall*

Current security mechanisms

• Hypercall(s) executed in a way such that
  • the targeted hypervisor cannot properly handle, or
  • an erroneous program code is executed.

Introduction  Analysis  Recommendations  Conclusions
Securing Hypercall Interfaces: Reactive Approaches

Workloads That Contain Hypercall Attacks

Evaluation of hypercall security mechanisms

“We have some difficulties to fully evaluate the efficiency of our hypercall protection measures…” Le (2009), pg. 47

Conclusions

Vulnerability Analysis

35 hypercall vulnerabilities analyzed
Reverse-engineering released patches → Proof-of-concept code

Observations

Implementation errors – missing value validations: Performance ↔ Security
Non-implementation errors are very common
   Are hypercall interfaces reliable?
Hypercall attacks: Effective DoS attacks, parts of multi-step attacks
Attack models: Mostly regular hypercalls
Conclusions

Recommendations

Securing hypercall interfaces: Proactive approaches
   Approaches and tools for fuzzing hypercalls
   Secure hypercall programming practices
   Methods for formally verifying hypercalls

Securing hypercall interfaces: Reactive approaches
   Novel hypercall attack detection techniques
   Approaches for generating workloads that contain hypercall attacks

Thank you for your attention
References


