Advanced OS Report

倪远东 2017210845

Virtual machine monitors: Current technology and future trends

Background

- 1960s:硬件资源昂贵
  - 硬件复用，诞生VMM技术
- 1980s:成本降低、多任务操作系统
  - VMM逐渐被淡忘
- Now:硬件成本降低，主机服务多易崩溃，管理困难
  - 在单台机器(或集群)上运行多个虚拟机，每个虚拟机提供特定服务
  - 资本涌入

硬件复用 -> 提供安全性、稳定性、便捷性保障

CPU虚拟化

- 概述
  - VMM技术使得虚拟机指令在CPU上可以“直接执行”
- 挑战
  - 2005年，没有CPU支持硬件虚拟化
- 技术
  - 半虚拟化，将“直接执行”与“二进制翻译”相结合
- 未来趋势
  - AMD和Intel相继推出硬件虚拟化支持

内存虚拟化

- 概述
  - VMM维护一个影子内存，用于映射虚 拟机中的内存数据
- 挑战
  - VMM没有足够的信息来管理活 跃分页
- 技术
  - GuestOS中运行Ballooning进程，弹性分配
- 未来趋势
  - Extended Page Table

I/O虚拟化

- 软件模拟
  - 有效复用，传输效率低
- 半虚拟化
  - 共享一块内存区域
- 直接分配
  - 最大化性能

总结

- Now
  - 安全性
  - 隔离性
  - 方便快捷
- Futrue
  - **应用分发的全新思路**

My VM is Lighter (and Safer) than your Container

Background

- 虚拟机
  - 稳定性高
  - 安全
  - 镜像大、速度慢
- 容器技术
  - 资源占用低
  - 启动速度快
  - 隔离性不足

Background

完美的LightVM在以下几点可以和容器媲美:

- 快速启动:容器可以毫秒级启动
- 高实例密度:一台主机上的容器数量级可以上万
- 暂停、恢复功能:快速暂停恢复

问题分析

为什么虚拟机相对于容器的性能如此低？

- 完整的系统，镜像比较大
- 运行的进程比较多

现象:用于应用分发，通常运行单个服务

改进:精简虚拟机中的软件，让其正好满足一个程序的运行环境

系统设计

测试Xen，找到瓶颈，进行改进

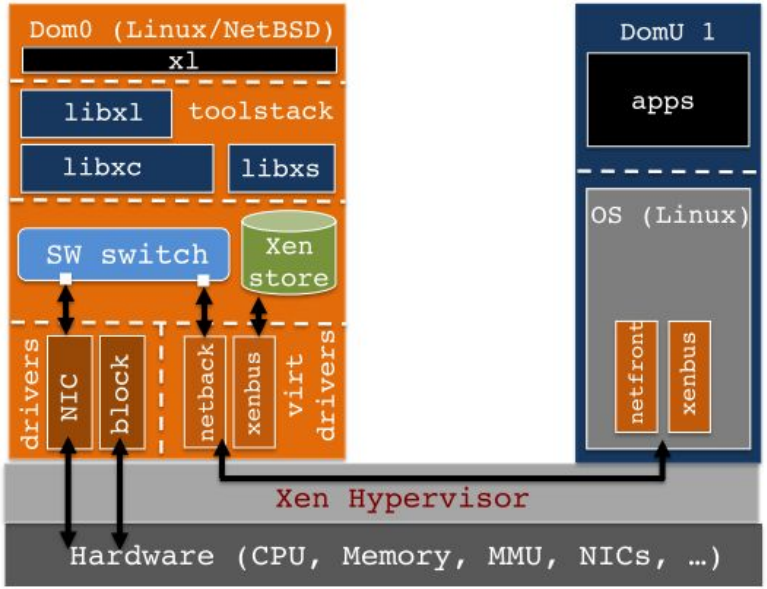


Figure 3: The Xen architecture including toolstack, the XenStore, software switch and split drivers between the driver domain (Dom0) and the guests (DomUs).

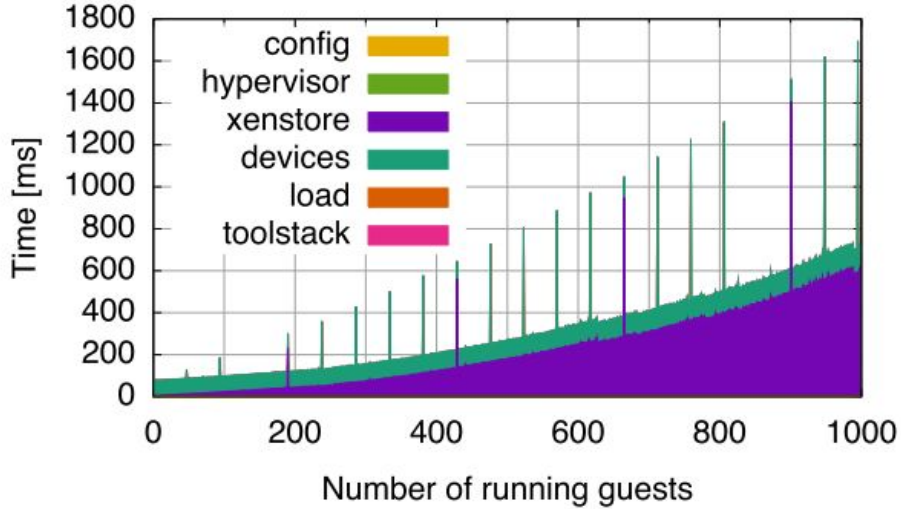


Figure 5: Breakdown of the VM creation overheads shows that the main contributors are interactions with the XenStore and the creation of virtual devices.

测试结果

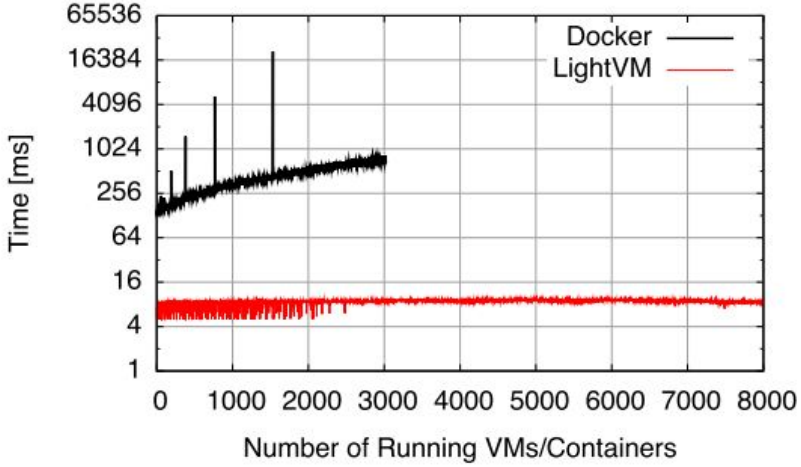


Figure 10: LightVM boot times on a 64-core machine versus Docker containers.

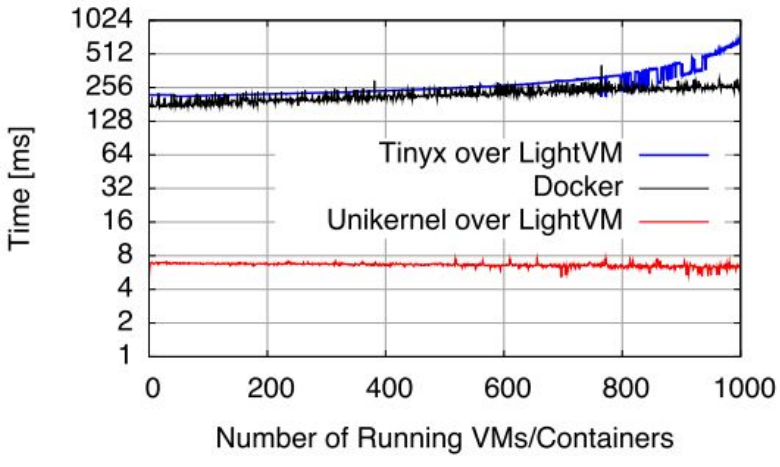


Figure 11: Boot times for unikernel and Tinyx guests versus Docker containers.

Making Smart Contracts Smarter

Background

- 比特币
  - 区块链诞生
- 以太坊
  - 区块链2.0
  - 智能合约

Background

- 基于以太坊的智能合约
  - 栈式虚拟机
  - 三种数据访问方式
    - 运行时栈
    - 内存
    - 长期存储

攻击介绍

- 交易顺序依赖
- 时间戳依赖
- 异常处理缺失
- 重入攻击

智能合约漏洞检测工具Oyente，采用符号执行进行分析

- CFGBuilder：为目标合约代码构建一个较为基础CFG图
- Explorer：遍历CFG，执行指令
- CoreAnalysis：漏洞分析核心模块
- Validator：消除误报

Oyente实验

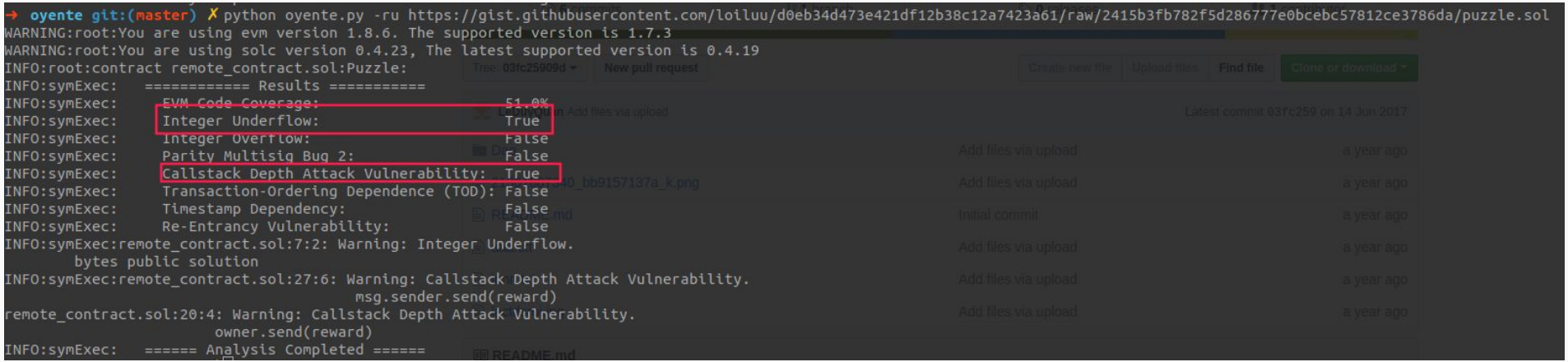
硬件：Intel i7 8550U/16GB RAM
操作系统：Ubuntu 18.04
语言：Python 2.7.14

- 搭建环境
- 安装依赖
- 下载合约文件
- 测试本地合约
- 测试远程合约

测试本地合约

```
→ oyente git:(master) ✗ python oyente.py -s ./source_code/DAO/DAO.sol
WARNING:root:You are using evm version 1.8.6. The supported version is 1.7.3
WARNING:root:You are using solc version 0.4.23, The latest supported version is 0.4.19
INFO:root:contract ./source_code/DAO/Token.sol:Token:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                  56.7%
INFO:symExec:     Integer Underflow:                  True
INFO:symExec:     Integer Overflow:                   False
INFO:symExec:     Parity Multisig Bug 2:              False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:               False
INFO:symExec:     Re-Entrancy Vulnerability:          False
INFO:symExec:./source_code/DAO/Token.sol:44:5: Warning: Integer Underflow.
    string public symbol
./source_code/DAO/Token.sol:43:5: Warning: Integer Underflow.
    string public name
INFO:symExec:   ====== Analysis Completed ======
INFO:root:contract ./source_code/DAO/TokenCreation.sol:TokenCreation:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                  59.5%
INFO:symExec:     Integer Underflow:                  True
INFO:symExec:     Integer Overflow:                   True
INFO:symExec:     Parity Multisig Bug 2:              False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:               False
INFO:symExec:     Re-Entrancy Vulnerability:          False
INFO:symExec:./source_code/DAO/TokenCreation.sol:42:48: Warning: Integer Underflow.
    /// @return Whether the token creation was successful
    ^
Spanning multiple lines.
./source_code/DAO/TokenCreation.sol:42:24: Warning: Integer Underflow.
    /// @return Whether the token creatio
INFO:symExec:./source_code/DAO/TokenCreation.sol:61:13: Warning: Integer Overflow.
            balances[_tokenHolder] += msg.value
Integer Overflow occurs if:
    balances[_tokenHolder] = 736057007607799978112749683587525671091669441922372553612328602841868119899209
./source_code/DAO/TokenCreation.sol:62:13: Warning: Integer Overflow.
            totalSupply += msg.value
Integer Overflow occurs if:
    totalSupply = 736057007607799978112749683587525671091669441922372553612328602841868119899209
INFO:symExec:   ====== Analysis Completed ======
→ oyente git:(master) ✗
```

测试远程合约


→ oyente git:(master) ✗ python oyente.py -ru https://gist.githubusercontent.com/loiluu/d0eb34d473e421df12b38c12a7423a61/raw/2415b3fb782f5d286777e0bcebc57812ce3786da/puzzle.sol
WARNING:root:You are using evm version 1.8.6. The supported version is 1.7.3
WARNING:root:You are using solc version 0.4.23, The latest supported version is 0.4.19
INFO:root:contract remote_contract.sol:Puzzle:
INFO:symExec: ============ Results ===========
INFO:symExec: EVM Code Coverage: 51.0%
INFO:symExec: Integer Underflow: True
INFO:symExec: Integer Overflow: False
INFO:symExec: Parity Multisig Bug 2:   False
INFO:symExec: Callstack Depth Attack Vulnerability: True
INFO:symExec: Transaction-Ordering Dependence (TOD): False
INFO:symExec: Timestamp Dependency: False
INFO:symExec: Re-Entrancy Vulnerability: False
INFO:symExec:remote_contract.sol:7:2: Warning: Integer Underflow.
bytes public solution
INFO:symExec:remote_contract.sol:27:6: Warning: Callstack Depth Attack Vulnerability.
msg.sender.send(reward)
remote_contract.sol:20:4: Warning: Callstack Depth Attack Vulnerability.
owner.send(reward)
INFO:symExec: ====== Analysis Completed ======

程序分析

- **oyente.py**：入口函数，可以接收以下类型输入
  - solidity合约代码
  - evm字节码
  - 远程合约代码
  - ***部署在链上的合约abi***
- **symExec.py**：执行分析的函数入口
  - build_cfg_and_analyze()
  - collect_vertices()、construct_bb()
  - full_sym_exec()
    - sym_exec_ins()

Precise and Scalable Detection of Double-Fetch Bugs in OS Kernels

Background

- 地址空间划分
  - 用户空间
  - 内核空间
- Multi-read 的普遍性
  - 依赖查找
  - 协议/签名校验
  - 信息猜测

Background

What is double-fetch bugs？

```
1  static int perf_copy_attr_simplified
2    (struct perf_event_attr __user *uattr,
3     struct perf_event_attr *attr) {
4
5    u32 size;
6
7    // first fetch
8    if (get_user(size, &uattr->size))
9      return -EFAULT;
10
11   // sanity checks
12   if (size > PAGE_SIZE ||
13       size < PERF_ATTR_SIZE_VER0)
14     return -EINVAL;
15
16   // second fetch
17   if (copy_from_user(attr, uattr, size))
18     return -EFAULT;
19
20   ......
21 }
22 // Example: if attr->size is used later
23 // BUG: attr->size can be very large
24 memcpy(buf, attr, attr->size);
```

(a) C source code

Overview

- 对double-fetch进行了正式和精确的定义
- 设计工具DEADLINE来自动化发现double-fetch
- 发现了Linux kernal中的23个新bug，以及FreeBSD中的1个新bug

形式化定义

1. 至少有两次对用户空间的读操作(multi-reads)
2. 两次读取内容必须有重叠部分(overlapped-fetch)
3. 两次读取数据必须有一定依赖（data/control)
4. 两次数据不能被证明一定是相同的

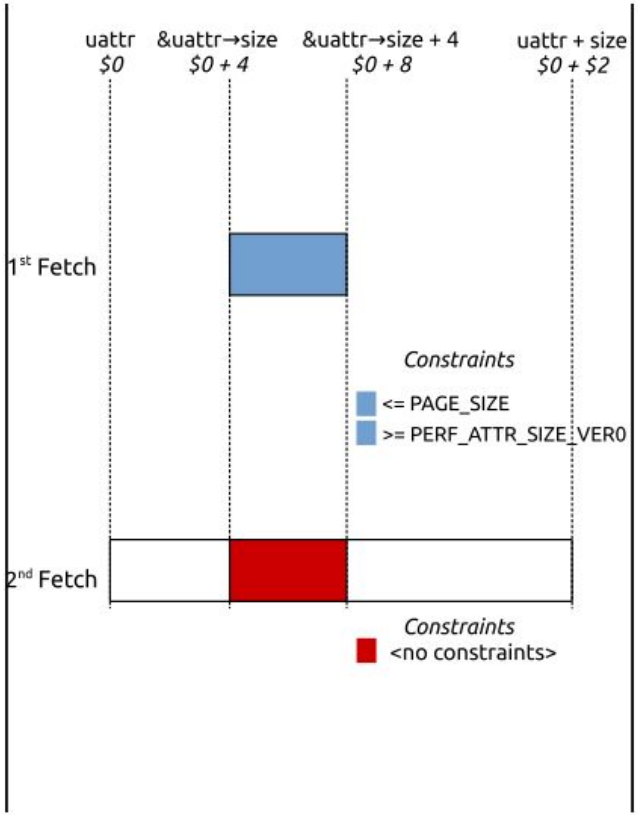
DEADLINE

Algorithm 1: High-level procedure for *double-fetch bug* detection

In : $Kernel$ - The kernel to be checked
Out : $Bugs$ - The set of *double-fetch bugs* found

```
1  Bugs ← ∅
2  Set_f ← Collect_Fetches(Kernel);
3  for F ∈ Set_f do
4      Set_mr ← Collect_Multi_Reads(F)
5      for < F_0, F_1, Fn > ∈ Set_mr do
6          Paths ← Construct_Execution_Paths(F_0, F_1, Fn)
7          for P ∈ Paths do
8              if Symbolic_Checking(P, F_0, F_1) == UNSAFE then
9                  Bugs.add(< F_0, F_1 >)
10             end
11         end
12     end
13 end
```

从multi-reads到double-fetch



(b) Memory access patterns

```
1  // init root SR
2  $0 = $PARM(0),     @0 = $UMEM(0) // uattr
3  $1 = $PARM(1),     @1 = $KMEM(0) // attr
4  ---
5  // first fetch
6  fetch(F1) is {A = $0 + 4, S = 4}
7  $2 = @0(4, 7, U0),@2 = nil       // size
8  ---
9  // sanity checks
10 assert $2 <= PAGE_SIZE
11 assert $2 >= PERF_ATTR_SIZE_VER0
12 ---
13 // second fetch
14 fetch(F2) is {A = $0, S = $2}
15 @1(0, $2 - 1, K) = @0(0, $2, U1)
16 ---
17 // check fetch overlap
18 assert F2.A <= F1.A < F2.A + F2.S
19     OR F1.A <= F2.A < F1.A + F1.S
20 // --> satisfiable with @0(4, 7, U)
21
22 // check double-fetch bug
23 prove @0(4, 7, U0) == @0(4, 7, U1)
24 // --> fail, no constraints on @0(4, 7, U1)
```

(c) Symbolic representation and checking

Hyperkernel: Push-Buton Verification of an OS Kernel

Background

- 内核的“正确性”尤为重要
- 如何证明内核设计的“正确性”
  - 形式化验证
- 如何自动化验证，提高可靠性和效率
  - Hyperkernal

难点

设计这样一个自动化工具面临着三大挑战

- 设计需要在可用性和自动化证明之间进行折中
- 分离内存中的用户空间与内核空间地址，简化证明中的内存管理
- C语言会使得推理复杂，如何建模

如何设计有限的接口：

- 强制资源被引用的生命周期
- 强制进行细粒度的保护
- 验证链接资源的结构

例：FD机制返回最小可用文件描述符

规范

用状态机来描述期望的内核行为：

- 抽象的内核状态规范
- 声明的规范

给出了两个定理，使用Z3进行定理验证

Hyperkernal实验

硬件：Intel i7 8550U/16GB RAM

操作系统：Ubuntu 18.04

- 搭建环境
- 安装依赖
- *make*
- *make qemu*
- *make hv6-verify*

```
    C++       irpy/compiler/irpy
    IRPY      o.x86_64/hv6/hv6.py
Parsing took 37.945 ms.
terminate called after throwing an instance of 'std::invalid_argument'
  what():  stoi
Aborted (core dumped)
Makefile:208: recipe for target 'o.x86_64/hv6/hv6.py' failed
make: *** [o.x86_64/hv6/hv6.py] Error 134
```

The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors

如何评价软件的可扩展性？

- 软件架构设计
- **接口设计？**

规则

如果一组操作在内存访问上是没有冲突的，我们就认为这组操作时可扩展的

- SIM交换性规则
- 可扩展性规则

设计可扩展的接口

设计可扩展接口的策略:

- 分解复合的操作
- 去掉接口中的一些并不需要的功能
- 弱化不必要的排序约束
- 异步释放资源

COMMUTER

- ANALYZER:自动分析接口的复杂性，给出交换条件
- TESTGEN：用于从ANALYZER给出的交换条件中生成大量的测试用例
- MTRACE：将测试用例运行在实际的系统上，并检查实现是否无冲突