2010 Second WRI World Congress on Software Engineering

Model Checking a Secure Hypervisor

Sunlv Wang, Jian Liu, Qiuping Yi, Xian Zhang
Institute of Software, Chinese Academy of Sciences, Beijing, P.R. China, 100190
liujian@iscas.ac.cn

***Abstract*—Hypervisor is a piece of platform-virtualization software that allows multiple operating systems to run on a host computer concurrently. CASMonitor, short for CAS Virtual Monitor, is a secure, high-assurance hypervisor prototype, which aims to level B3 or higher of TCSEC standard. This paper reports our experience of employing model checking method to verify some design properties of CASMonitor, such as isolation, mediated sharing, communication between separated virtual machines and source control policy. We show how to specify design architecture of CASMonitor with Spin PROMELA language and verify the above important properties to meet system security request.**

***Keywords*-Model checking; Secure operating system; Hypervisor; Chinese-Wall policy;**

I. Introduction

CASMonitor [7] is a secure and high assurance hypervisor prototype derived from the open source Xen hypervisor, whose target is the level B3 or higher level of TCSEC standard [3]. CASMonitor can be regarded as micro-kernel which encodes in about 55000 loc of C/ASM, divided into domain control, scheduling and security control modules etc.

Spin is one of the most popular model checking tools that used for formally verifying the design properties of systems. This paper reports our experience on using Spin to automatically verify design model of CASMonitor. In our model, we formally define the main functions of major modules in CASMonitor, including scheduling, hypercall, memory management, event channel, hardware interface, security control and some user applications. In the security controlling module, a Chinese-Wall control policy [2] is equipped, which is used as labels interpreted to decide which domains can co-exist on the same system.

This paper is organized as follows. In Section 2, we describe the CASMonitor together with its Spin model. In Section 3, we discuss the specification and verification of the important properties including Chinese-Wall correctness, consistency and isolation. In Section 4, we show the related work, and finally conclude this paper.

II. The Model

The following subsections explain the details of five subsystems in CASMonitor respectively, which are domain control subsystem, scheduling, Memory, Hypercall, Chinese Wall. For each subsystem, we first give the description of its main functions in CASMonitor, then we give the Spin model.

A. Domain Control

Domain is the most important concept in CASMonitor, which can be regarded as a ”super process” in classical operating system to control the life-cycle of guest OS and hold resources for its running. Domain is implemented by a piece of code and described by a data structure *domain descriptor* [5]. A descriptor structure contains an `id` which is the unique identifier of a domain, a level of privileges that used to control the memory access and scheduling [1], a context containing the state data, status information for scheduling, and a set of other specific data (for example, event channel for communication and secure identifier for access control etc).

Besides domain, the interrupt handler is a special control-flow, which manages the soft and time interrupt and interaction between hyperviosr and domain.

Spin Model. We use the Spin processes to model the domain and other control-flows. The Spin processes can execute interleavedly to simulate actual execution. In this setting, each control-flow such as domain is given a unique Spin process id which is the context recorded in descriptor. Each descriptor is represented by a PROMELA data-structure contains the Spin process id, the scheduling information (defined in Section II-B), the message queue, the event channel for communication, specific data for Chinese-Wall policy, and an array to store local data.

B. Scheduling

As we mentioned before, CASMonitor is an Xen based hypervisor architecture and the scheduling is the same as Xen. In this paper, the round rubin algorithm is selected to model as in Xen 2.0.

Spin Model. Scheduling is activated after an interrupt operation is finished. We suppose the system runs on a single-CPU machine. This means, only one domain or hypervisor is allowed to run at a time. Base on this suppose, we implement a priority-based round-robin scheduling algorithm. Hypervisor has the highest-priority (RING 0) and all the domains have the same priority (RING 1) that lower than hypervisor's. The idle process has the lowest-priority, and it is a special process executed when no hypervisor or domain is ready.

978-0-7695-4303-1/10 $26.00 © 2010 IEEE
DOI 10.1109/WCSE.2010.115

IEEE computer society

For each domian,there are three different statuses: RUNNING, READY, and BLOCKED. Surely, if the hypervisor or domain is scheduled to run, its status must be READY. For this purpose, a new field `schedInfo` is introduced to store the scheduling status and priority. When a scheduling event occurs, the scheduling queue is traveled, and the scheduling decision is stored in a global variable `curr_id` that indexes a descriptor. Then, the status information of selected process will be changed to RUNNING.

C. Memory

CASMonitor is responsible for managing the allocation of physical memory to guest OS, and scrubbing free memory. Besides allocating and scrubbing memory, virtual memory managing is also a basic memory service. In fact, virtualizing memory provides mechanisms such as protecting guest domain from each other and translation between address spaces, which is the most difficult part of so-called para-virtualizing architecture.

Due to the limited description of PROMELA, we simplify the memory manger services as follows: CASMonitor partitions memory into independent regions to each domain associated with a memory access privilege level. The access is granted if the execution privilege is higher, or the access is denied if the execution privilege is equal or lower.

Spin Model. We model the memory access mechanism by an array that associates each region with its privilege level and a set of operate functions that models memory accesses. Before each memory access action, the access function checks the current execution privilege (`CPU_MODE`).

In our model, array `mem` is the descriptor of all domains Array `used` is used to keep track of which descriptors are allocated. The initialization function `hm_init` sets all the descriptors' state (record in array used) to initial values. The allocation function `hm_alloc` tries to find an unallocated descriptor and returns the index of the descriptor, or the return value is assigned to -1 if the array is full. The deallocation function `hm_free` sets a descriptor's state of a given index to free.

D. Hypercall

Hypercall is a software trap from a guest OS to hypervisor. Domains use hypercalls to execute privileged operations. When a domain makes a hypercall and raises an interrupt, which passes control from RING 1 to RING 0, i.e. from domain running privilege to hypervisor's privilege.

To use this mechanism, a domain sets its register to appropriate values (service types and arguments), raises a software interrupt, and switches the running context form RING 1 to RING 0 to activate an interrupt handler. According to the service type, hypervisor transfer the control flow from the guest OS code to the corresponding manipulation function.

Spin Model. In PROMELA, a channel variable `chan` can be used to transfer messages between active processes. We use a global channel to pass arguments from a domain to the interrupt handler, whereas the response is sent back to the domain through a private channel whose pointer is passed as the reply arguments of the message receiving and sending functions [5]:

```
inline recv_msg(from, smsg, reply) {...};
inline snd_msg(to, smsg, reply) {...};
```

The main function of hypervisor is implemented as a infinite loop to receive messages from domains through handler function. If no message comes, the hypervisor's status is changed to BLOCKED and declare waiting for a message. Otherwise, hypervisor looks up the message arguments and processes the interrupt.

The hypervisor executes the hypercall code, and uses sending function to send the return value back to the domain. Similar to hypervisor, domain uses receiving and sending functions to raise interrupt and receive feedback, and changes CPU MODE.

E. Mandatory Access Control

As a secure hypervisor, CASMonitor integrates a flexible mandatory access control subsystem to control communication between domains and isolate domain. A Chinese Wall policy is implemented into this module. The policy defines a group of chwall-type, usually a chwall-type referring to a certain data set used by a guset. And then, a conflict set is defined as a set of chwall-types [6]. Each domain is assigned a Chinese Wall label consisting of a set of chwall-types for this domain. Hypervisor starting a domain is according to the following rule: if two domains' labels are in a same conflict set, then these two domains cannot run simultaneously.

Spin Model. We use an global array, `conflictset`, to maintain current running domain and conflict information. Each element in the array stores a domain's Chinese Wall label `ssidref`. In this setting, when Hypervisor wants to create a new domain, it first calls the function as follows.

```
inline chwall_pre_create(ssidref, i)
{...};
```

This function reads the label of `ssidref` field in domain's descriptor, and the argument `i` is used as a return value. If a previous domain at the same conflict is already built and still running, `i` is assigned to -1, otherwise hypervisor calls the creation function to start the new domain, adds the domain's label to the conflict sets, and stores this domain id. In addition, when a domain is destroyed, hypervisor calls `chwall_domain_destory` function to clean the corresponding label in conflict sets.

III. EXPERIMENTS

In this section, we present several verifications done on formal specification model of CASMonitor described in the previous sections: the correctness of the Chinese-Wall policy, the consistency of the hypercall and the isolation property.

A. Chinese-Wall Correctness

According to our model, the application to create a domain has two outcomes: success with the user request's state updated to SUCCESS, or return failure and the corresponding conflict set was already occupied by another guest OS. We use following LTL formula to describe this property.

```
#define request
  (_usr_request[_curr_req].state ==
  STATAPPLY)

#define wait (appstate == WAIT)

#define cre_failed
  (_usr_request[_curr_req].state == FAILED
  && _user_domain[conflictsets[
  _usr_request[_curr_req].type].domid])

#define cre_ok
  (_usr_request[_curr_req].state ==
  SUCCESS)

[](request -> wait U (cre_failed || cre_ok))
```

When a user wants to create a new domain, request's state is assigned to `STATAPPLY`. In the specification, `cre_failed` means the user failed to create a domain, and the `cre_ok` expresses the opposite outcome. As we mentioned before, the `domid` field of `conflictset` structure stores id of the domain which occupies this conflict set. If a creation request is failed, this outcome also implies that one of the mutual exclusive domains is running. In our model, we use an array `_user_domain[]` to trace each domain's status, so we can look up this array to determine whether a mutual exclusive domain is running.

B. Consistency

The consistency property states [5] that a domain waiting for a message (raise a hypercall) can never be scheduled. Before the expected message received, the domain's scheduling status must be BLOCKED. We describe this property as follows.

```
#define domainrun
  (_curr_ctxt == (mem[curr_id].contextPtr))

#define waitdomain
  (_user_domain[init_domain_id] &&
  mem[init_domain_id].msgQueue.
  msgPendingStatus == WAITING)

#define blockdomain
  (_user_domain[init_domain_id] &&
  mem[init_domain_id].schedInfo.status
  == BLOCKED)

[]((waitdomain && domainrun) ->
  blockdomain)
```

In this formula, `init_domain_id` is the user domain id, and `domainrun` ensures that the current running process is a domain not the handler. If the executing process is the handler, the property is invalid.

C. Isolation

The isolation property is important for a CASMonitor system, and this property means all the guest OS can only run on RING 1 level. This property implies that whenever a domain is running, it have no right to access the memory of hypervisor which is in RING 0 privilege level. Only the hypervisor code can access protected memory area, and if a domain want to read or write it, a hypercall must be called. That is to say, domains must use hypercall to communicate or access protected areas. We define `userdomainrun` and `kernellevel` assertions to represent this property. These two assertions' details are similar to the former definitions, so we give the LTL formula directly.

```
[](userdomainrun -> !kernellevel)
```

D. Results

In order to increasing reasonable, our whole model contains 780 lines PROMELA code for hypervisor, and 150 lines for applications. The experiment runs on a Core 2 Duo 2.00GHz machine with 16GB of RAM. We set the Spin verification option Maximum Search Depth to 10,000 to avoid incomplete state search and select the "Use Compression" optimization option to reduce memory consumption. We measure resource consumption in function of the different number of domains created in our model. Moreover, the verifications of "consistency", "isolation" use almost the same amount of resource, so these two outcomes are merged into one graph.

Figure 1 details the resource consumption in verification including memory consumption and time consumption. We find these three verifications' consumption increase exponentially during the experiment. In the third verification (3 domains), Spin reports valid result after running for around 280 seconds with a maximum memory utilization of more than 8GB. We attempted to add a forth domain, but the verification exceeded the memory capacity of the machine. From the results, we also find the verification of "correctness" uses more resource. That is mainly since the

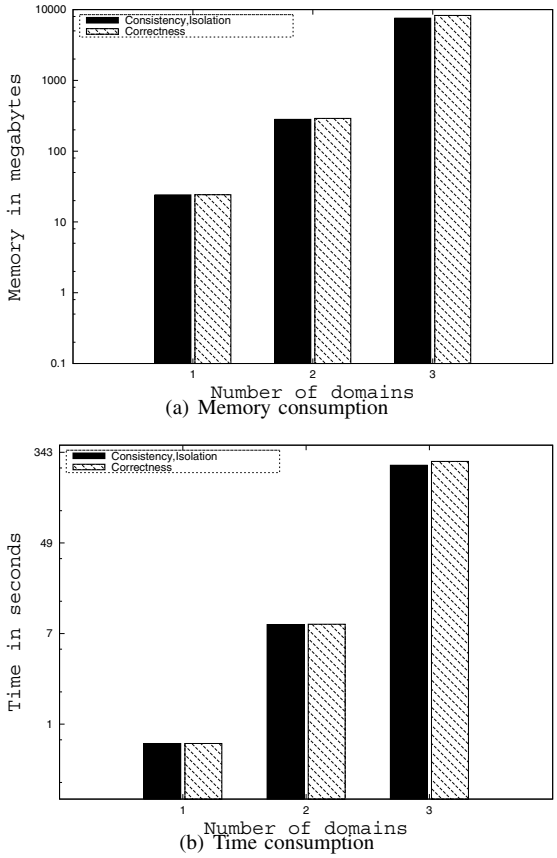

Figure 1. Resource consumption in verification, including memory space and execution time.

two temporal modalities nested in the LTL formula require an additional traversal of the state space.

IV. Related Work

Our control flow and memory model are inspired by the work of Nicolas Marti et al. who also use Spin to check some import properties of Topsy [5], a tiny embedded OS. Their work is building in the Spin model-checker a model of Topsy and verifying high-level properties, while our focus is on security policy verification of Hypervisor.

In Nicolars's paper, their aim is the proving of high-level properties of the embedded system. However, in our work, we are more interested in the separation and security control of CASMonitor than others. Moreover, this work is just part of big project about development of CASMonitor and more detail specification and verification will carry out in future work.

Jason Fanklin et al. have also analysed the design of a secure hypervisor named Secvisor [4] with model checking method. A formal specification of the memory protection scheme of SecVisor is created with Murφ in their work, and moreover, they report finding and repairing some security defects of SecVisor in the paper. In contrast, our work is based on a complete model of hypervisor, including the memory subsystem, the hypercall and other main components.

V. Conclusion

In this paper, we specify and verify the design of CASMonitor using Spin model checker. Our work not only provide a complete design model of the system, but also concern their important secure properties. As a secure and high-assurance computing platform, the formal model contains the classical parts of a operating system. Moreover, we integrate the Chinese-Wall policy into our formal model. Based on this model, we verify the separation property between different domains, which is important to a high-assurance system.

Our experiment shows a effective high-assurance method to development of micro kernels, and it reasonable enough to support our future work and apply on the similar systems.

Availability

The PROMELA code and three LTL formulas are available from http://code.google.com/p/securehypervisor/

Acknowledgements

This paper was supported in part by National High Technology Research and Development Program of China (863 Plan) under grant No. 2009AA010313 and the Key Project of Chinese Academy of Sciences under grant No. KGCX2-YW-125.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[2] D. F. C. Brewer and M. J. Nash. The Chinese Wall security policy. *IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[3] Department of Defense. *Trusted Computer System Evaluation Criteria (TCSEC)*. DoD 5200.28-STD.

[4] J. Franklin, A. Seshadri, N. Qu, S. Chaki, and A. Datta. Attacking, repairing, and verifying secvisor: A retrospective on the security of a hypervisor. Technical Report CMU-CyLab-08-008, Carnegie Mellon University/Cylab, 2008.

[5] N. Marti, R. Affeldt, and A. Yonezawa. Model-checking of a multi-threaded operating system. In *23rd Workshop of the Japan Society for Software Science and Technology, University of Tokyo, Tokyo, Japan*, 2006.

[6] J. M. McCune, S. Berger, R. Cáceres, T. Jaeger, and R. Sailer. Shamon: A system for distributed mandatory access control. In *Proceedings of the Annual Computer Security Applications Conference*, pages 23–32, 2006.

[7] B. Z. Wu. Casmonitor: A os monitor based on hypervisor framework. Master's thesis, Graduate University of Chinese Acadmy of Sciences, Beijing, China, 2010.