Poster: Fast Object Identification for Kernel Data Anomaly Detection

Hayoon Yi, Yeongpil Cho, Donghyun Kwon, Yunheung Paek
Dept. of Electrical and Computer Engineering, Seoul National University
hyyi, ypcho, dhkwon, ypaek@sor.snu.ac.kr

***Abstract*—As recent adversaries turned their eyes to attacking a system through non-control kernel data, in order to ensure the integrity of the kernel, the need arose for verifying non-control kernel data. This complicates typical security measures relying on *integrity specifications* set by security administrators, as it is non-trivial to manually encompass specifications for non-control kernel data. Foreseeing this, Baliga et al. [1] suggested a framework leveraging machine learning to generate integrity specifications with little human involvement. Unfortunately, there is a problem in the original design of this framework in regards to its practicality for deployment in real-world systems. In this paper, we propose a new design in identifying kernel objects that accelerates the overall introspection process by virtually eliminating the booting delay that was needed in prior work.**

I. Introduction

In efforts to seek sophisticated techniques to compromise OSes, adversaries have turned to tampering with non-control kernel data. Unfortunately, as typical security systems rely on *integrity specifications*, which describe the behavior expected from an uncompromised kernel, set by security administrators, the need to encompass specifications for non-control data brings forth a major problem: it is virtually impossible to provide hand-crafted specifications for all non-control data in the kernel to verify the kernel integrity. Acknowledging this, Baliga et al. [1] suggested a framework named Gibraltar that leverages machine learning to generate specifications with little human involvement. Unfortunately, despite the advantages of Gibraltar, there is a major problem in its practicality. Gibraltar employs externel monitors, which are isolated, physically or virtually, from the potential influence of contaminated kernels, performing *memory introspection*, the act of looking into and making sense of the raw memory of a different system. The data objects of the target system are found by their relative position to public symbols, whose addresses are predetermined at compile time. These relative positions, however, are subject to change after a system reboot, and thus any specification related to an object with an alternate position must incorporate its new position in order to correctly examine its current runtime value. Consequently, Gibraltar must track down data objects and infer their invariant properties at the start of each and every reboot of the system, which takes up from 20 to 50 minutes even on an up-to-date machine.

In this paper, we propose a new design leveraging information available at object allocation events, namely, backtraces of kernel function calls, to identify objects persistently over reboots, substantially cutting the time needed at reboots as well as the time needed to verify specifications at runtime. To evaluate the effectiveness of our design, we have implemented a prototype data anomaly detection engine. Preliminary experiments reveal that we need only a delay of 68.49ms with each reboot and a delay of 912ms for anomaly scanning.

NDSS '17, February 26 - March 1 2017, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2017.23xxx

II. Motivation

In memory introspection systems, kernel data objects must first be identified within raw memory. However, the monitor initially has little information of the actual kernel data residing in memory. To overcome this, the monitor typically leverages known semantic information of data objects and maps out the data objects within raw kernel memory. During this process, each object is given a *name* to distinguish them from one another. For the generation of integrity specifications, the invariant properties of the mapped data objects are inferred from their found data values, be it by hand or machine learning, and associated with their corresponding data objects. The association of object name and invariant property would be reflected in integrity specifications. Then, during runtime, these specifications are checked for data anomaly detection. In such a system, there exists a complication, namely, *transient integrity specifications*, which are specifications that hold true during the runtime they were inferred but may not hold true across system reboots. There are mainly two reasons for these specifications: the object naming used for specifications being transient or the inferred invariant property being transient. In this paper, we focus on the former. As mentioned before, Gibraltar identified kernel objects by recursively following member pointer fields of data structures and there is no guarantee that these relative positions would persist over system reboots. This is because one or more objects on a pointer traversal path may be of a container type of data structure, say a linked-list, and in this case, the path is subject to change after every system reboot, as the contents of the container cannot be decided until runtime. This renders many generated specifications to be transient. The authors of Gibraltar argued that persistent specifications alone were sufficient enough to detect all of the attacks from their test suite. However, we discovered that there are attacks that cannot be covered by their set of persistent specifications. For example, VFS (Virtual

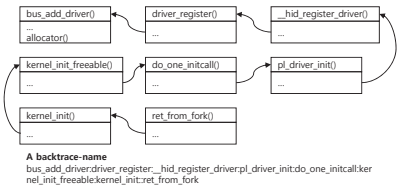


Fig. 1. An example backtrace-name with its corrsponding function call trace

File System) [2] rootkits are a well-known class of rootkits hijacking control flows during file management. They attempt to manipulate member function pointers of file objects which designate the functions to be called when file operations such as read and write are carried out. In order to detect this class of rootkits, the value of the *f_op* variable inside file objects must match a value in a list of legitimate *f_op* values. The path-name for file objects, however, are subject to change over reboots, therefore only with transient specifications (of Gibraltar) can a VFS rootkit be detected.

III. Approach

We propose *Backtrace-naming*, which is naming objects with their allocation backtrace, a backward list of active function calls that starts with the last function call at the time of allocation. In other words, whenever an allocation occurs in a function, the object is named with the backtrace of that function, which would represent the context of the object's creation. For instance, Figure 1 depicts a backtrace of the kernel functions calling *bus_add_driver* and the corresponding backtrace-name for the object allocated by the allocator function. This name would hold true for the object under any circumstance, barring a change in kernel code. Consequently, backtrace-names would persist over reboots, allowing us to reuse integrity specifications generated during a one-time offline inference. The basis for the use of backtrace-naming for kernel integrity monitoring is the following two observations: **O1.** Kernel objects are allocated through only a couple of fundamental object allocators, **O2.** The kernel context when a kernel object is created reflects the object's characteristic during runtime. Whenever an object allocation event occurs, we gather relevant information through the virtual machine manager (VMM). Although there are various types of object allocators in the kernel, according to our examination, most are nothing but wrapper functions that ultimately call fundamental object allocators, as in **O1**. For example, the *alloc_task_struct_node function*, which allocates *task_struct* objects, internally calls the *kmem_cache_alloc_node* function; in other words, it wraps the other function. Likewise, *kmalloc* is a wrapper function of the *kmem_cache_alloc* function. Therefore, we can gather most object allocation information of kernel objects by only tracking a couple of fundamental allocators.

Unfortunately, backtrace naming cannot give every individual object their own unique name and a few objects are bound to share the same name, and thus the same integrity specifications. Our observation **O2**, however, gives insight that objects created in a similar kernel context would have similar characteristics at runtime, and thus their sharing of specifications is not necessarily a bad thing in this case. For instance, *inodes* are kernel objects used in various kernel functionalities, such as file systems, sockets or device drivers.

The number of allocations	186,132
The number of live objects	29,765
Avr. CPU cycles per trap	321
Avr. CPU cycles per backtrace-naming	140
Total spent CPU cycles of traps	116,440,503
Total spent time(ms) of traps at 1.7GHz	68.49

TABLE I. Overhead for object identification and naming during kernel boot

They are allocated by the function *alloc_inode* which is a wrapper function for *kmem_cache_alloc*, which we have found that could be called through 259 distinct backtraces. The reason for this is that, instead of generating and distributing inodes from a central component, each kernel component generates their own inode in accordance to their distinct kernel context. As a result, inodes sharing the same backtrace-name show similar characteristics, and thus the specifications shared among them reflect the properties based on these similarities.

IV. Preliminary Results

We implemented a prototype system leveraging backtrace naming on an Arndale board, with an ARM Cortex-A15 1.7 GHz dual-core processor and 2 GB RAM, supporting hardware virtualization extension. The prototype is integrated to a KVM with Linux and is based on Gibraltar with the exception of employing backtrace-naming for object naming. Backtrace naming is enabled by hypervisor traps. Table I reports information on a normal kernel boot. Anomaly scanning after bootup requires an average of 912ms which is a huge decrease from that needed in the original Gibraltar design as well.

V. On-Going Work

We plan to optimize the runtime monitoring phase by shortening the scanning time by reducing the amount of kernel data checked each time. We are also planning to analyze the automatically generated specifications and find a way to automatically refine the specification set by eliminating redundant or unnecessary specifications.

Acknowledgement

This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2014R1A2A1A10051792), Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. R0190-16-2010, Development on the SW/HW modules of Processor Monitor for System Intrusion Detection)and Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. R-20160222-002755, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning).

References

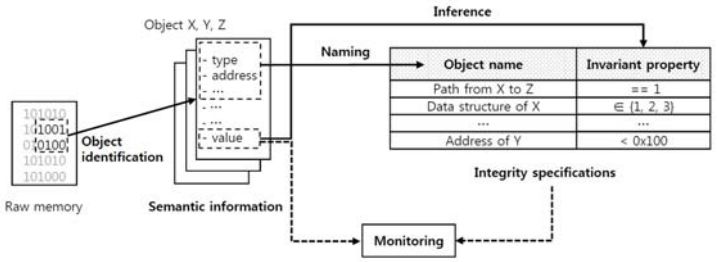
[1] A. Baliga, V. Ganapathy, and L. Iftode. Automatic inference and enforcement of kernel data structure invariants. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, pages 77–86. IEEE, 2008.

[2] D. P. Bovet and M. Cesati. Understanding the linux kernel, 2 ed. In *Advances in Cryptology–EUROCRYPT 2014*. OReilly and Associates, Dec, 2002.

Fast Object Naming for Kernel Data Anomaly Detection

Hayoon Yi*, Yeongpil Cho*, Donghyun Kwon*, Yunheung Paek*
*Seoul National University

Memory Introspection



Motivation

- Deployed security systems usually rely on integrity specifications, which are typically set by a security administrator
- Non-control data attacks in kernel

➔Need for kernel data integrity

- Unfortunately, it is nontrivial to manually set specifications for all kernel data

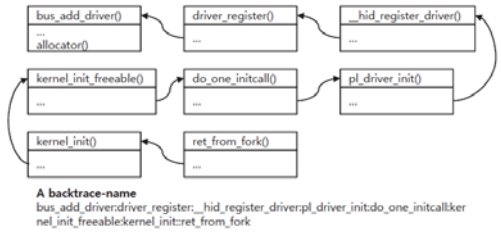
➔ Automated specification generation with machine learning

- Prior work was done in this area
  - A.Baliga, V.Ganapathy, and L.Iftode. Automatic inference and enforcement of kernel data structure invariants. ACSAC 2008
- Has an issue that a large portion of generated specifications not being applicable after a system reboot

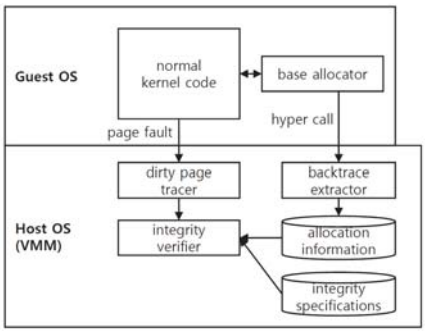
➔Needs to re-generate specifications after each reboot, which takes 20~50minutes even on an up-to-date machine

Backtrace Naming

Backtrace: a backward list of active function calls that starts with the last function call
Naming: Giving an identity to a data object to distinguish it from other objects with different names



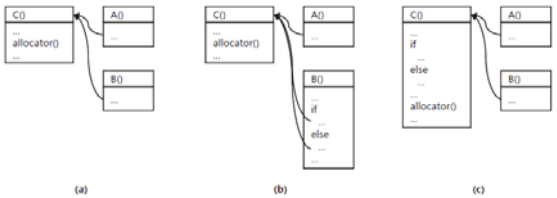
Prototype Overview



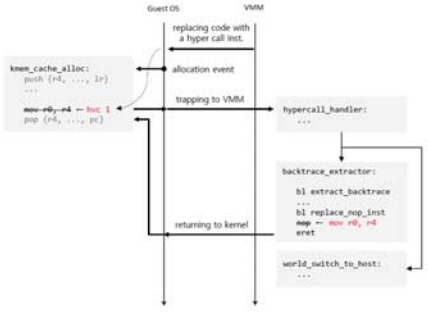
Naming Granularity

How would different naming schemes effect the granularity of which data objects would be distinguishable

- Path-name
- Type-name
- Backtrace-name



Backtrace extraction at an allocation event



Key Observations

1. Kernel objects are allocated through only a couple of fundamental object allocators.
2.The kernel context when a kernel object is created reflects the object's characteristic during runtime.

Preliminary Experiments

The number of allocations	186,132
The number of deallocations	156,367
The number of live objects	29,765
Avr. CPU cycles per trap	321
Avr. CPU cycles per backtrace-naming	140
Total spent CPU cycles of traps	116,440,503
Total spent time (ms) of traps at 1.7GHz	68.49

Acknowledgements

This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2014R1A2A1A10051792), Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. R0190-16-2010, Development on the SW/HW modules of Processor Monitor for System Intrusion Detection)and Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. R-20160222-002755, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning).