An enhancement of trusted domain enforcement using VMM interruption mechanism

Ruo Ando
National Institute of Information and Communication Technology
4-2-1 Nukui-Kitamachi, Koganei Tokyo 184-8795 Japan
ruo@nict.go.jp

Youki Kadobayashi
National Institute of Information and Communication Technology
4-2-1 Nukui-Kitamachi, Koganei Tokyo 184-8795 Japan
youki-k@nict.go.jp

Yoichi Shinoda
National Institute of Information and Communication Technology
4-2-1 Nukui-Kitamachi, Koganei Tokyo 184-8795 Japan
shinoda@nict.go.jp

Abstract

TDE (Trusted domain enforcement) is an extension of TE (Type Enforcement) for dynamic access control and sandbox. VMM (Virtual machine monitor) provides a strong isolation which can enhance TDE of guest domain. In this paper we propose an enhancement of trusted domain enforcement of guest domain using VMM interruption mechanism. VMM based TDE is achieved by two steps. A non-write protected input for guest domain is detected, notification is sent to VMM. Then, VMM isolates the domain from other domains. Proposed system needs notification channel passing VMM and guest OS modification. We discuss the way to create the notification channel by a newly added VMM interruption. Also, the modification of guest OS is presented. We implement input validation routine by kernel patch and LSM (Linux Security Module) and compare CPU utilization.

1 Introduction

1.1 Virtual machine monitor

VMM (virtual machine monitor) is a thin layer of software between the physical hardware and the guest operating system. The rapid increase of CPU performance enables VMM to run several operating system as virtual machine, multiplexing CPU, memory and I/O devices in reasonable processing time. Recent VMM is a successful implementation of microkernels. Under the guest OS, VMM runs directly on the hardware of a machine which means that VMM can provides the useful inspection and interposition of guest OS.

1.2 Security of VMM

VMM is also called hypervisor. Hypervisor is not new. VMM now offers huge potential to generic servers about consolidation and secure isolation [1]. Classical aspects of secure kernel for VMM is discussed in [2][3]. Recently, XEN [4][5] and KVM (kernel virtual machine) [6] is going to mainline of linux kernel.

To obtain system control, security module need to be at lower-level of system. Secure OS is a kernel module located between system call execution and I/O access. With the deployment of VMM under operating system, some security applications of VMM has been proposed. The design and deployment of IDS on VMM is discussed in [7]. They propose VMI (virtual machine introspection) applied for IDS policy management. Some of the LKM-rootkit is tested to validate their system. In [8][9], moving functionality of secure Linux module to outside guest VM is discussed. However, in these papers memory forensics to extract further information of malware is not discussed explicitly.

In another advantage of VMM, VMM provides new con-
cepts for verification and debugging. These are called as deterministic replay [10] or time-travel debugging [11] which is also useful for security. Security application of logging and replay using VMM are discussed in [12]. However, their paper is not specified for the detection of kernel mode malware such as LKM-rootkit. Computer forensics is much concerned with rootkit detection and inspection. VMM based rootkit is discussed on [13]. In [13], they do not mention how to detect the event of installation of VMR. The thrust of this paper is the specific modification of VMM architecture in order to enhance memory snapshot for malware forensics. Besides detection, proposed system enables us to extract further information of exploitation of kernel based malware. Once the split kernel module is implemented and registered correctly, modification of guest is simple. Also, evidence memory block can be extracted by a simple string analysis.

1.3 Secure OS and sandboxing

Recently the exploitation of vulnerability for compromising system such as buffer overflow have been serious problem. For preventing or minimizing damage, sandbox and secure OS have been proposed. SELinux[17], Apparmor[18], Trusted (Open) Solaris[19] are now popular and widely used. These systems have three common features: MAC(Mandatory Access Control), TE(Type Enforcement), and RBAC(Role based access control). Trusted path execution is proposed for preventing malicious code execution. However, if the program such as bash and perl passing TPE[20] check is compromised, we cannot inspect the path of execution. In this paper we introduce TDE (Trusted Domain Execution) for input path validation.

Sandbox is a restricting program execution into protected area of storage and memory. It prevents "bad effect" of program for outside the protected area. Most widely known sandbox is Java sandbox which is cope with Java byte code. Classical sandbox for native code is Janus[21] and TRON[22]. Windowbox[23] creates the isolated desktop on Windows 2000. Virtual machine application such as VMWare[24] and UML[25] can be applied for sandbox. Chroot, a system call for virtualizing fs namespace is applied for constructing sandbox. Jail[26] in FreeBSD isolates process and network interface. In this paper we cope with VM based sandbox approach.

1.4 VMM based TDE

We have implemented TDE on Xen Virtual Machine Monitor as shown in Figure 1. VMM provides strong isolation which can enhance TDE of guest domain. Our VMM based TDE needs two steps: If an unprotected input for guest domain is detected, guest domain is isolated by disabling virtual network interface by Xen domain controller (XEND).

2 Trusted Domain Enforcement

TDE (Trusted Domain Enforcement) is an idea of dynamic sandbox. It is similar to TE (type enforcement) with input validation. TE and TPE has a limitation when some programs such as interpreter has been compromised. For example, if attackers hijacks bash, he can misuse bash by using unprotected files as input, which is not detected by access control. The main concept of TDE is as follows:

The process that reads unprotected input is forced to move in untrusted domain with its capability removed (even if the process is privileged).

TDE is implemented in LIDS (Linux Intrusion Detection System) for kernel 2.4 series. TDE is not implemented on kernel 2.6.x. In this paper we have implemented TDE for Linux kernel 2.6.x for dynamic sandboxing with VMM interruption mechanism.

2.1 Input validation

TDE has an inspection of input file for process about write protection. Non-write protected is regarded as unprotected (input). When the unprotected input for the process is detected, capability (permissions) of process are all removed. That is, the process is sandboxed. For example,

Subject ACCESS inherit Object
----------------------------------
test WRITABLE 0 /tmp

Under this ACL, one the process accesses (has input) of tmp directory, the capability is removed. The process cannot use resources including executables in /bin, /sbin and so on. In the view of inspecting attributes of the process, TDE is similar to TPE. But TPE including sandboxing for the process.

2.2 Sandbox: from blacklist to whitelist

When the state of the guest domain moves from left side to right side, access control list is changed from black list to white list. All capabilities of the guest domain is removed and access right need to be granted explicitly. In black list based, we need to prohibit access explicitly. But after sandboxing, every possible (necessary) access of the guest domain needs to be listed as ALLOW. For example, normal Linux system has DENY based ACL. we put DENY for objects to access such as

Subject ACCESS inherit Object
----------------------------------
test WRITABLE 0 /tmp

Under this ACL, one the process accesses (has input) of tmp directory, the capability is removed. The process cannot use resources including executables in /bin, /sbin and so on. In the view of inspecting attributes of the process, TDE is similar to TPE. But TPE including sandboxing for the process.
Figure 1. Trusted Domain Enforcement (sandboxing) of guest domain using bridge controller (virbr) of Xen. When an unprotected input is detected on guest domain, TDE is activated (TDE=1) as stopping bridge (virbr0) by Xen controller (Xend).

Subject ACCESS inherit Object
----------------------------------
Any file DENY: 0 /etc/shadow
Any file APPEND: 0 /var/log

With trusted domain enforcement, ACL is changed to as follows:

Subject ACCESS inherit Object
----------------------------------
sandboxed READONLY 0 /bin/commandA
sandboxed READONLY 0 /lib/libA

If command A is granted for sandboxed object, we need to allow access of necessary libraries for command A.

2.3 TDE implementation in Linux

We add tde_enabled flag to structure task_struct to check whether current process is under TDE or not. Then, if the process is TDE enabled, we validate the input for the process by modifying sys_read() / sys_write(). When the violation is detected, we notify it from kernel space of guest OS to VMM by asynchronous notification mechanisms. Trusted Domain Enforcement is implemented in LIDS (Linux Intrusion Detection System) 1.1 series. 1.1 series is running on Linux 2.4.x which is implemented as kernel patch. Kernel code around fs directory (file system) is modified, for example, sys_read() / sys_write(). In this paper we have implemented it for Linux 2.6.x on VMM (including sparse kernel).

3  Building notification channel using VMM interruption

TDE is dynamic resource access control. Therefore VMM based TDE needs asynchronous interruption for control domain to isolate guest domain dynamically. In this section we discuss the way to construct notification channel using VMM interruption.

3.1 Event-channel of XEN

We have implemented proposed system on XEN virtual machine monitor. In XEN, asynchronous notification channel between guest domain and control domain is called event channel. Event channel is virtualized software interruption. I/O request of guest domain is sent from frontend driver to backend driver via event channel as shown in Figure 3. In proposed system, we newly construct event channel for asynchronous notification.
3.2 Adding event channel

In proposal method, memory snapshot need to be taken timely. Polling or sequential mechanism is not proper to cope with the security incidents. Particularly, on-line or asynchronous mechanism is necessary for inspecting malware on memory. In proposal system we construct an asynchronous notification channel using split kernel module (device driver). When security incidents such as buffer overflow exploitation, malware installation and malicious file access has been occurred on the guest OS, it is notified to our frontend device driver. Then, the notification is transferred to backend device driver. Device driver is more suitable (than application) for coping with the asynchronous events.

In modified VMM, the detected incidents are equivalent of a hardware interrupt. When malicious behavior is detected, it is notified to our kernel module as IRQ (interrupt request). Let us show the list of the guest OS.

<table>
<thead>
<tr>
<th>IRQ</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>256: 1782</td>
<td>Dynamic-irq timer0</td>
</tr>
<tr>
<td>257: 0</td>
<td>Dynamic-irq resched0</td>
</tr>
<tr>
<td>258: 0</td>
<td>Dynamic-irq callfunc0</td>
</tr>
<tr>
<td>259: 227</td>
<td>Dynamic-irq xenbus</td>
</tr>
<tr>
<td>260: 187</td>
<td>Dynamic-irq xenscons</td>
</tr>
<tr>
<td>261: 891</td>
<td>Dynamic-irq blkif</td>
</tr>
<tr>
<td>262: 0</td>
<td>Dynamic-irq blkif</td>
</tr>
<tr>
<td>263: 0</td>
<td>Dynamic-irq eth0</td>
</tr>
<tr>
<td>264: 0</td>
<td>Dynamic-irq sec-notify</td>
</tr>
</tbody>
</table>

This is the list of dynamic (virtualized) IRQs of the guest OS. We have obtained this list by "cat /proc/interrupt". 261 is interrupt handler of block device. 262 is for swap. 263 is for network interface. 264 sec-notify is our frontend driver of split kernel module. We append dynamic-irq sec-notify and by using this, the incident notification is transferred to VMM.

Once our driver is registered correctly, modification of guest OS is very simple. All we need to do is inserting this code into probes of guest OS.

```c
int port=9;

evtchn_op_t op;
op.cmd = EVTCHNOP_send,
op.u.send.port = port;
(void)HYPERVISOR_event_channel_op(&op);
```
This code is activating event-channel and sending the interruption signal to our frontend-drivers in port 9. In this case, port number 9 is assigned to Dynamic-IRQ 264. By inserting this code, the incident notification is transferred to frontend driver, VMM, backend driver and finally the host OS.

The deployment of proposed system with split kernel module is completed in these steps.

1. Implementing front/backend driver (split kernel module)
2. Registering driver to Xenstore
3. System call extension: inserting hypervisor call into system call (create\_module)
4. MAC extension: inserting hypervisor call into i-node permission checker

Xenstore in (2) is the utility of XEN, device database of the guest OS. By step (3)(4) and (5), secure OS and other protection system can communicate with our split device driver. We discuss the step (3) and (4) in the next section. We discuss step (5) in appendix section. These steps are completed only by inserting five lines of code above.

4 Guest OS modification

In this section we discuss TDE implementation for guest OS. There are two ways to implement TDE in Linux kernel 2.6 series, kernel patch and LSM. In both ways, In kernel patch, we modify the files in fs (file system) and kernel directory. In LSM, we cope with LIDS (Linux Intrusion system)[27] as example in modifying i-node permission check routine.

4.1 Flowchart of proposed method

Figure 4 show the flowchart of trusted domain enforcement. For TDE, we need to check two points. First, if the process has TDE Flag or not. That is, it is going to be under TDE or not. For checking these points, we extend (adding member) the structure of task\_struct and d\_entry.

4.2 Implementation of kernel patch and LSM

For making kernel patch, we modify the files in fs (file system) and kernel directory. In LSM, we cope with LIDS for modifying i-node permission checking routine. In both cases, we add the variable TDE\_enable into struct task\_struct in sched.h to set TDE flag for the process.

```c
struct task_struct {
```
To activate TDE for target process, we insert additional routine into do_execve of exec.c and do_fork of fork.c For example,

```c
int do_execve
{
    file = open_exec(filename);

    -- TDE check routine using file->f_dentry structure --
    task->tde_enabled = 1;
    /* TDE enabled */
}
```

Input validation routine is inserted into sys_read of read_write.c The routine checks if input are write-protected or not using structure file->f_dentry.*:

```c
asmlinkage ssize_t sys_read
{
    task = current;

    -- input validation using file->f_dentry --

    /* if this process is with TDE enabled ? */
    if(task->tde_enable==1)
    {
        /* TDE activated. Capability removed. This domain is sandboxed. --
    }
}
```

In the case of input validation of LSM, the additional code is inserted into inode permission check function. For example,

```c
static int lids_inode_permission
(struct inode *inode, int mask, struct nameidata *nd)
{
    task= curent;

    --Input file is write-protected or not ? --

    /* if this process is with TDE enabled ? */
```
if(task->tde_enable==1) {  
    -- TDE activated.  
    Capability removed.  
    This domain is sandboxed. --  
}  

Figure 5 shows the comparison of CPU utilization of Kernel patch and LSM. We generated read-events in the non-write protected (under TDE) and measure CPU system time using vmstat. As long as we experimented, utilizations makes much difference. However, utilization increases faster in kernel patch because of LSM is additional operation layer between kernel and device drivers.

5 Conclusions

TDE (Trusted domain enforcement) is the extension of TE (Type Enforcement) for dynamic access control and sandbox. VMM (Virtual machine monitor) provides a strong isolation which can enhance TDE of guest domain. In this paper we have proposed an enhancement of trusted domain enforcement of guest domain using VMM interruption mechanism. VMM based TDE is achieved by two steps. A non-write protected input for guest domain is detected, notification is sent to VMM. Then, VMM isolates the domain from other domains. Proposed system needs notification channel passing VMM and guest OS modification. We have discussed the way to create the notification channel by a newly added VMM interruption. Also, the modification of guest OS has been presented. We have implemented input validation routine by kernel patch and LSM (Linux Security Module) and compare CPU utilization. VMM based TDE provides the powerful isolation and simple architecture which can be implemented by a few hundreds of lines of code. For further work, proposed system could be ported to other OS and VMM.

References


[24] VMWare is available at: http://www.vmware.com


[27] Linux Intrusion Detection System: http://www.lids.org/