# Sevigator: Network Confinement of Malware Applications and Untrusted Operating Systems

Keywords: Virtualization-based Security, Network Access Control, Hypervisor, Virtual Machine Monitor, Virtualization, Security, Privacy Protection

Abstract: Stuxnet worm opened a new era in cybersecurity. This heavily networking virus implemented a new threat: it infects industrial control systems; consequences of the infection might be as gravy as a technogenic catastrophe. Stuxnet performs networking to communicate between instances, contact so called "Stuxnet command and control center", and upload malicious code to real-time controllers. The virus uses OS exploits to infect a computer and installs its component in OS kernel, gaining full control over infected node. This paper presents Sevigator – a toolkit for network confinement when only trusted application gain access to local network while other application and even OS kernel have no networking at all. Thus Sevigator effectively prevents Stuxnet scenario. Sevigator is based on hardware virtualization support: a custom hypervisor hides network interface card from the OS kernel and delegates network-based system calls of trusted applications to a dedicated service virtual machine. To prevent code injection or data alteration by a malicious kernel code or driver the hypervisor maintains integrity of the trusted applications binaries, shared objects and in-memory data.

## 1 INTRODUCTION

On June 17, 2010 one more virus was identified by antivirus community virusblokada(Nicolas Falliere, 2011). While the report looked routine, it opened a new era in cybersecurity: it was the first instance of Stuxnet worm .Stuxnet is a multifunctional software with blocks responsible to penetrate into MS Windows environment, infect computers in local network, setup P2P network for distributed self-upgrade, and, most notably, modify control loop of programmable logic controllers (PLC) of industrial control systems. The latter function makes Stuxnet unique among all other viruses its ultimate target is control systems. To disrupt the control process it installs properly signed drivers into infected Windows hosts and then uses network to upload malfunctioning software to PLC.

It is worth noting the role of the network in Stuxnet infection. The virus uses network to communicate between instances of the worm, get updates from so called "command center", and uploads malware to PLC. Traditional firewalls provide only partial protection from Stuxnet and alikes, since they do not know exact context for IP packets — whether the sender is a trusted process or a malware, whether the contents of the packet is valid or it is altered by a malicious software. This statement holds for both bump-in-the-wire firewalls that run on physically separate computer or router, and firewalls running on the same network node with the malware. The reason is that modern monolithic OS grant low-level software virtually unbounded rights. If a malware managed to compromise OS and installed a driver deep into the kernel, it can either mimic a system process or even inject malware code into the running trusted application.

In this paper we present Sevigator – a hypervisor-based systems to protect network access when the operating system is compromised with malware. The goal of Sevigator is to provide network access to trusted applications only, ensuring confidentiality and integrity of the traffic: malware has no possibility to send, receive or alter network traffic, even if it resides deep in a monolithic kernel.

The approach is based on the hardware virtualization facilities(Adams and Agesen, 2006) available in the modern families of AMD and Intel CPUs. With virtualization enabled, OS is loaded after a special program called hypervisor. Hypervisor privileges exceed the privileges of the operating system; hardware support gives hypervisor full control over computer resources. The original intention of hypervisor support was minimization of virtualization overhead, but the facilities of the hardware virtualization support lead to other applications as well.

Virtualization support was introduced in AMD and Intel CPUs in 2005-2006. In the following years a number of protection approaches were published, some of them required changing the software(Garfinkel et al., 2003)(Ta-Min et al., 2006) or

even hardware(Dwoskin and Lee, 2007)(Lee et al., 2005). Despite performance gains those approaches are hardly usable for seamless transparent protection of existing systems.

One of the most advanced approach to application protection from a compromised OS is Overshadow(Chen et al., 2008), developed by a research team from leading US universities and VMware company. The key point of the approach is memory encryption when a trusted application is interrupted and memory decryption when the control returns. As a result, no external process, including OS kernel and drivers, can read plaintext application's memory. Encryption is implemented by hypervisor. Similar approach is proposed in (Yang and Shin, 2008).

Due to utilization of hypervisor Overshadow can be used to protect applications without modifications. It targets perfect integrity and confidentiality of application data. Still but one can point out that:

- encryption and decryption of large memory chunks is an expensive procedure in terms of CPU utilization and time;
- it gives partial protection against Stuxnet-like attacks where malware has access to network interface card and can issue arbitrary packets to the network.

Sevigator is close to Overshadow but imposes less overhead for protection and provides fine-grained access to networking facilities. It is based on a hypervisor that:

- confines network interface card from untrusted applications, including OS kernel. Only trusted applications can perform network exchanges;
- ensures integrity of trusted applications' code and data by means of secure hash codes. Discarding encryption makes task switch less expensive in terms of CPU utilization and time.

Combination of these two factors gives confidentiality of application data and protection from unauthorized network transaction. To prove confidentiality just note that even a malware managed to spoof in-memory data of an application it is unable to report it to criminals or malware instances on other network nodes.

## 2 ARCHITECTURE

In Sevigator the hypervisor ensures simultaneous execution of two virtual machines (VM), primary and service, completely isolated from each other. The primary VM contains secured applications; it is the VM that potential users have access to. The service VM is designed to process network input/output for trusted application of the primary VM. It includes the minimal set of applications and might run a dedicated verified operating system. The selection of the service OS in production environment is guided by the need to minimize the risk of exploits and system compromise through networking attacks.

Still by now we used both VMs running the same untrusted operating system (OS) in order to simplify networking facility. It is worth noting that the requirement for the OS to be the same is not severe. The abolishment of this requirement does not lead to the expansion of the critical code, i. e. the code, the vulnerabilities in which determine the security of the whole system. The current requirement results from the early stage of the project implementation.

A user works in the primary VM, which controls all devices except for the network adapter, through which the information leakage might occur. The primary VM is granted strictly one processor (processor core). It includes critical data, and programs handling this data are executed in it (both trusted and untrusted). The data are stored in a plain (unencrypted) form, and protection system does not limit read access of processes (both system and user's) to these data making its processing possible by any programs including untrusted ones.

When primary VM starts the hypervisor blocks its access to the network interface. An OS running in it believes a network adapter to be physically absent. Therefore any attempt to establish a network connection from within this VM and transfer data to the remote computer will unavoidably lead to an error. Hence the malicious code running within the primary VM with any level of hardware processor privileges will not be able to transfer critical data, even though it manages to get access to it.

Among all user programs included in the primary VM, a number of trusted programs requiring communication with remote network computers for their proper operation is distinguished. The service VM is used to provide network access to them. This VM is running in a background mode. It should be mentioned that the only peripheral device served by the service VM is a network adapter. Any application within tis VM can interact with remote network computers. Because of VM isolation provided by the hypervisor software in the service VM can not get access to the data contained in the primary VM.

Network adapter driver and TCP/IP stack could be implemented in the hypervisor. The use of the supplementary VM leads to additional expenses for processor time and computer memory. However the re-

quirement of minimality of trusted code is a priority requirement. It also provides a possibility to maintain the modularity of the protection system. The use of new VMs for servicing an access to other peripheral devices such as hard drive is possible. Description of the hard drive protection goes beyond the scope of this paper, though the overall concept is the same as network access protection presented here. The third VM ensures files integrity rather than information confidentiality. A separate hard drive is assigned for its exclusive control, with read access provided to all processes and write access provided only to trusted ones. This VM constitutes a trusted virtual machine and is running under Minix3 OS. The operating principles of the third VM in the frame of the security system being considered coincide with those described further related to the system calls servicing and data communication between VM and the hypervisor.

The support of network communication for the trusted applications is implemented by means of remote servicing of the required set of system calls in the service VM. The only data communicated outside the primary VM are explicitly specified by the arguments of a trusted process system call with the transfer of data outside this VM being carried out by the trusted code (the hypervisor).

The trusted processes are executed under the untrusted OS. In the absence of proper control by the security system, the malicious code in the OS kernel can load required code into the address space of the trusted process, pass control to it, and the trusted process in its own name will execute all the actions on the critical data delivery to the remote computer controlled by intruder. To avoid such a malicious effect, security system protects the context of the trusted application from an unauthorized modification by any code in the primary VM, including privileged code.

# 3 PROTECTION OF THE TRUSTED APPLICATIONS CONTEXT

Malicious software in OS kernel has capabilities to alter application's executable file or modify the shared libraries that the application uses. As a countermeasure we perform transparent validation of the application's executable file and its shared libraries when a trusted application is being launched. Authenticity is validated by means of secure control sums of the pages of the memory-mapped file of the application or a library. Checksums are 160-bit SHA-1 hashes, one for each page of static data and code in a binary file. Checksum failure for at least one hash leads to the loss of the trusted status of the application. The procedure of checksum verification is implemented in the hypervisor. System administrator has ability either to hardwire checksums for every trusted application and library into the hypervisor at the compilation stage or to dynamically load them into the hypervisor during the system operation.

Besides the libraries specified in the executable, an application may dynamically load libraries. The hypervisor intercepts mapping of those files into memory and validate it.

Dependencies on the libraries, which can not be extracted explicitly from the trusted application executable file, shall be specified by the administrator at the moment of its checksum read-out. There is a possibility to selectively provide trusted status to the application context. For example, if an application requires use of a library which for some reason cannot be executed in the trusted context, the possibility exists of abolishment of the trusted status for the period when execution thread is passed from the application to the above mentioned library with further status restoration.

Another criterion for trusted application is command line parameters used to launch the application. System administrators may specify that the application acquire trusted status for certain command line only.

An integrity control of the execution thread is carried out by means of address tracking of executable file start instructions, OS return instructions and instructions of control passing to the signal handlers. As this takes place, the general-purpose register state is also controlled.

Every trusted process is executed in a separate security domain. It is essential for the current status maintenance of control tables and assuring access interception to the desired memory pages. Security domain is nothing more than a dynamically changeable set of physical pages with specified read and write access rights. An attempt of the executable code to get access to the physical page outside its domain as well as an access to the page within its domain accompanied by the access violation is intercepted by the hypervisor.

The implementation of security domain technique is based on the use of an independent set of nested VM registration tables for each domain (NPT implementation in the Intel processors is called Extended Page Tables, in the AMD processors — Rapid Virtualization Indexing). Nested tables specify translation of nominal physical VM addresses to the actual absolute physical addresses. The hypervisor creates a new

(empty) set of nested tables for the process when it enters he trusted mode for the first time. Every time the control is passed to the process, the hypervisor sets the nested tables for this process as active for the virtual machine. Upon the interruption of process execution and control pass to the OS, the hypervisor toggles active nested tables and sets the tables of an untrusted domain (domain of kernel and untrusted processes).

# 4 THE HYPERVISOR AND VM DATA COMMUNICATION METHODS

Remote servicing of the system calls is implemented by the hypervisorin association with the system components, working in both VMs, primary and service. During the system initialization modules are dynamically loaded into the kernel of the primary and service VM.

Each module allocates continuous physical memory space for ring buffer (RB), registers several interrupt handlers which are used by the hypervisor to notify virtual machine of incoming events or processing, and communicates this information (buffer address and interrupt numbers) to the hypervisor through a hypercall(AMD, 2011). A hypercall is a VM program call to the hypervisor for the purpose of execution of some action. Synchronous nature of this call makes it possible to transfer hypercall parameters, much as the user process transfers parameters to the OS kernel during the system call execution: numerical parameters and memory area addresses are passed through the registers, the hypervisor reads virtual machine memory area with the specified addresses and fetches addition information from there (or records it there) when needed.

The most complicated situation arises when the hypervisor has to notify a virtual machine of some event. To do this, the hypervisor uses a capability to throw interrupts and exceptions into the virtual machine by means of the corresponding fields in the VM control structure (AMD, 2011) provided by the virtualization hardware. Upon VM resumption, the hardware ensures interrupt delivery immediately before the execution of the first VM instruction. In response to the interrupt OS passes control to the corresponding interrupt handler which was registered in the interrupt handlers table by the kernel module during the system initialization process.

System call parameters are passed through the ring buffer. The buffer constitutes a circled array of data structures (of a fixed size); its head shifts as requests are fetched from the buffer and its tail shifts as requests are loaded into the buffer. If the buffer is overflowed, the request delivery is suspended until space in the buffer is freed up.

The data structure representing a ring buffer element is common for all system calls and includes fields for all possible fixed-length parameters. Variable-length parameters are transferred through the separate variable-sized buffer allocated in the hypervisor memory, storage. The coordinates of a variable-length parameter, an offset from the storage origin and a length, are specified in the data structure of the ring buffer. The hypervisor maintains a separate storage copy for every trusted process.

Upon the receipt of a request containing variable-length parameters, the VM code for which this request is intended performs a hypercall for the storage access sending a requested parameter coordinates and an address of the buffer in its own memory intended for holding the data from the storage. The hypervisor services the call and resumes VM execution. In doing so it controls that the scope of the requested block doesn't go beyond the storage. The storage may be accessed both for reading and writing.

# 5 REMOTE SERVICING OF THE NETWORK SUBSYSTEM SYSTEM CALLS

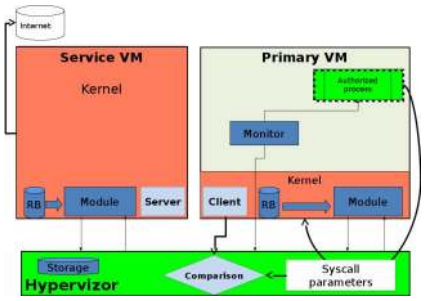


Figure 1: Configuration of the security system components in charge of remote execution of the system calls, and simplified chart of their interaction.

User applications use socket technique for the network accessing. One of the parameters specified during creation of a new socket is a protocol type. This parameter defines which handlers shall be called by the OS kernel for correct completion of a system call and creation of a new socket. In the subsequent data reading, writing, and other actions with the created socket the OS kernel handlers related to this protocol are called.

To ensure confidentiality of trusted applications

data untrusted applications and OS kernel in the primary VM are deprived of network access: when kernel enumerates PCI bus the hypervisor intercepts all replies from network card. As a result, the primary VM gets no knowledge about available network facilities and runs in network-less mode. For application it means that only loopback adapter is available, no ways to send or receive data from outside of VM.

Network access to trusted applications is provided by means of a dedicated stub in the kernel that routs all network requests to the service VM through hypervisor. To make the data routing transparent to application the system call socket is intercepted by hypervisor: it substitutes family parameter with the ID of the stub protocol that redirects all requests to the hypervisor. The stub protocol is implemented using standard procedures of Linux kernel. Protocol code can be divided into client and server parts. The server part runs in the service VM, and the client part runs in the primary VM (Figure 1).

When a system call from the trusted application arrives, the execution of the primary VM is interrupted and control is passed to the hypervisor before this call is handled by the primary VM kernel. The hypervisor substitutes a protocol identifier in the arguments of the system call by a special one corresponding to the new protocol, which comprises a part of the security system. The hypervisor also copies the parameters of system call from the trusted application. Then the execution of the primary VM is resumed. System call is received by the OS kernel. It analyses the system call parameters and passes control to the registered handlers in the client part of the new protocol.

The client part allocates required kernel resources, creates packet containing system call arguments, writes it to the ring buffer and notifies the hypervisor through the hypercall.

Once a control is passed back to the hypervisor, it partially compares previously stored system call arguments with those transferred through the ring buffer of the primary VM. Since the primary VM kernel is considered to be untrusted, such a comparison is required to avoid data leakage to forged addresses and ensure data integrity. In case of comparison error the received system call will not be executed in the service VM.

The hypervisor writes received data into the ring buffer of the service VM and throws an interrupt to it. Interrupt handler passes the control to the server part of the protocol. The data from the ring buffer are then unpacked. Then the standard handling of a system call with the received arguments is performed, as if this call was received from the user application running in the service OS without security system. The track presented above results in socket creation and IP packets been sent out by the service VM for a trusted application.

Processing of inbound packets is performed in the reverse order. The proxy application in the service OS passes received UDP or TCP data to the hypervisor via a hypercall. The hypervisor stores the received data into the ring buffer of the primary VM and throws an interrupt into it. The interrupt is handled by the stub protocol that reads the received data from the hypervisor and writes it to the proper memory regions allocated by the application and passed as parameters to the system call. When OS returns control to the trusted application by sysexit instruction, hypervisor intercepts it and checks that the data was not altered by the untrusted OS.

The complexity of returning results of the remote system call handling arises because some system calls can legally modify process memory. The hypervisor guarantees that only explicitly specified in the system call parameters address ranges will be modified. It is accomplished by the fact that during execution of the system call OS kernel driver allocates memory area of the required size and modifies the system call parameters, OS writes results in this memory area. The hypervisor copies the data into the corresponding memory area of the trusted process upon the return from the system call.

# 6 IMPLEMENTATION

Currently, the protection system is implemented as extension of KVM hypervisor. KVM runs in a host operating systems and uses x86 emulator QEMU for device emulation. According to (Goldberg, 1973) it is hypervisor of the second (hosted) type. In our work, we use KVM 88 version, QEMU 0.15 version and Linux kernel of 2.6.32 version. Primary and service virtual machines run on the kernel with the same version as host's one.

## 6.1 Performance

In order to measure overhead of remote system call execution and memory protection, we have performed several tests. In each measurement there was only one trusted process. The table below presents results of the tests for a few popular applications. Percent value is the extra time spent by the trusted application compared to the time spent by the same application with protection system disabled.

Table 1: Measurements.

| Software | Description | Overhead |
|---|---|---|
| Apache | Flood test. | 2% |
| Ttcp | Ordinary execution. | 2% |
| SSH&SCP | SCP 4Gb file copy. | 22% |

# 7 CONCLUSION

This article presents an approach to protect application confidentiality from untrusted (potentially compromised) OS. The approach is implemented by selectively granting a network resource access to the individual trusted user applications. Confidentiality is achieved by preventing untrusted, malware or compromised OS components from communication channels to the outside world. Besides confidentiality the presented approach provides efficient protection from Stuxnet-like attacks since it enforces a fine grained firewall bundled with thorough integrity protection.

The implementation of the approach is based on the execution of the untrusted OS within a virtual machine and placement of the trusted part of the security system in the hypervisor body. It allows achieving full control of access of the processes executed under VM to the hardware resources. The security system therewith remains inaccessible for attacks by malicious software.

The use of hardware virtualization technique provides two facilities:

- protection of integrity for trusted application, preventing malicious code injection or data modification;
- delegation of servicing of network-related system calls to the dedicated service system.

Facilities of system call intercepting, reading their parameters, and writing syscall results provided by virtualization makes it possible to protect access to any peripheral hardware resources, including hard drives.

The possibility to grant to individual processes a controlled access to the resources not accessible by the OS itself, provides a way to effectively solve particular issues of the user data confidentiality preservation problem and confines untrusted applications, including the OS kernel, within the network node, making malicious networking impossible.

At the moment the implementation of the hypervisor is based on KVM for Linux on AMD platform and it supports network confinement for Linux only. Future research goals are extension of the approach to Intel VT platform, support MS Windows family of operating systems, and "bare-metal" execution of the hypervisor.

# REFERENCES


(2011). *AMD64 Architecture Programmer's Manual Volume 2: System Programming*. Advanced Micro Devices Inc.

Adams, K. and Agesen, O. (2006). A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA. ACM.

Chen, X., Garfinkel, T., Lewis, E. C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., Dwoskin, J., and Ports, D. R. (2008). Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA. ACM.

Dwoskin, J. S. and Lee, R. B. (2007). Hardware-rooted trust for secure key management and transient trust. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 389–400, New York, NY, USA. ACM.

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., and Boneh, D. (2003). Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 193–206, New York, NY, USA. ACM.

Goldberg, R. P. (1973). *Architecture of virtual machines*, pages 22–26.

Lee, R. B., Kwan, P. C. S., McGregor, J. P., Dwoskin, J., and Wang, Z. (2005). Architecture for protecting critical secrets in microprocessors. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 2–13, Washington, DC, USA. IEEE Computer Society.

Nicolas Falliere, Liam O Murchu, E. C. (2011). W32.stuxnet dossier. Technical report, Symantec Corporation.

Ta-Min, R., Litty, L., and Lie, D. (2006). Splitting interfaces: making trust between applications and operating systems configurable. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 279–292, Berkeley, CA, USA. USENIX Association.

Yang, J. and Shin, K. G. (2008). Using hypervisor to provide data secrecy for user applications on a per-page basis. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, pages 71–80, New York, NY, USA. ACM.