Elevated Penetration Attack Models of Virtual Machine Escape Based on FSM

Wei Fan[1,*], Weiqing Huang[1]

[1]Institute of Information Engineering, CAS

Abstract

Virtual machine escape is one of the most serious vulnerabilities happening if the isolation between the hosts and between the VMs is compromised,which presents new security challenges that the security concern is the major factor effecting virtualization technology widely adopted in IT industry. In VM escape, the program running in a virtual machine is able to completely bypass thehypervisor layer, and get access to the host machine. The traditional research method is analyzing a vulnerability separately, but that consumes too much timeand not constructs the attack model. So we innovatively design VM escape elevated penetration attack models based on finite state machine, which could be used to identify potential vulnerabilities in design, implementation and testing phases. In this paper, firstly, we extract elevated privilege models of different virtualization methods, studying that VMCS pointer instruction state indicates system state. Secondly, we define a formal language Datalog to represent pre- and post-conditions of the exploits of application vulnerabilities and infer a basic elevated penetration attack model. Thirdly, through the analysis of vulnerable source code and vulnerability reports from NVD, we shed light on four attack models to cover the most VM escape attacks. Finally, we evaluate the presentedapproach by applying code-level finite state machine models with formal language to specific vulnerabilities, together with the statistical results of different attack models.

Keywords: Virtual machine escape, finite state machine, Datalog, elevatedpenetration attack model

Received on 14 November 2020, accepted on 23 December 2020, published on 08 January 2021

doi: 10.4108/eai.21-7-2021.170555

1. Introduction

Virtual machines share the resources of the host machine but still can provide isolation between VMs and between the VMs and the host. That is, virtual machines are designed in a way that a program running in one virtual machine cannot monitor or communicate either with programs running in other VMs or with the programs running in the host. But in reality the VM escape attacks have been already introduced tocompromise isolation, which are considered to be one of the worst case in virtualization system [1]. Since the host machine is the root, the program which gets access to the host machine also gains the root privileges basically and escapes from the virtual machine privileges. This result in complete breakdown in the security framework of the system environment. Moreover, it is expected that the exploitation of unknown VM escape vulnerabilities is increasing sharply. There is a pressing need for effectiveapproaches to identify and fix vulnerabilities, which is always tough and expensive. Modeling attack models not only contribute to reasoning potential vulnerabilities in testing phases, but also can guide the practitioners, especially system engineers with little knowledge of security to improve system security in design and implementation phases. An attack model is the abstraction of the basic property of an attack to identify how vulnerabilities may be exploited. The existing attack models either do not (1) consider vulnerable code when attack is modeled and then do not provide enough in- formation for mitigation strategies; or (2) consider too many or too few attack pattern categories which contribute little to understanding vulnerabilities and attacks.
This paper innovatively presents a basic VM escape attack model and four evolutional attack models, which are generalized based on an exhaustive research of National

[*] Corresponding author. Email: fanwei@iie.ac.cn

Vulnerability Database (NVD). Code-level finite state machine (FSM) paradigm [2] is used to model the attack patterns. Our contributions are summarized as follows:

We extract elevated privilege models of different virtualization methods. Through the research of the virtualization mechanism, it is inferred that VMCS pointer instruction state indicates system state.

We generalize four attack models by analyzing vulnerability reports from NVD. These models could be used to analyze both known and unknown vulnerabilities. Thesemodels filter the details of exploitation and abstract the key steps, thus facilitating the identification of the unknown vulnerabilities by matching system behaviors from code to these attack models.

A formal language is defined, which extends Datalog [3] to specify the pre- and post-conditions of transitions. By using reasoning logic based on the formal language,the description is simplified and formalized which benefits the deduction of potential vulnerabilities.

Code-level FSM models for the attack models are proposed, which apply the de- fined formal language to describe the condition/action of each transition. The models could describe attack scenarios clearly and formally. The capabilities of the FSM models are evaluated by applying them to specific vulnerabilities reported recently in NVD, together with the statistical results of different attack models.

The rest of paper is organized as follows. Section 2 introduces related work. Section 3 proposes problem statement. In Section 4, a formal language is defined and a basic attack mode is designed. In Section 5, different attack models are identified and modeled by code-level FSM. In Section 6, case study for the attack model is presented. Section 7 concludes our work.

2. Related Work

This section presents the existing research on security model and attack model analysis.Many studies have been carried out for security models. There are mainly two kinds ofapproaches for modeling vulnerability exploitation [3]. The first one is to analyze databy statistical methods and then use stochastic models to describe exploitations [4]. At-tack countermeasure tree is one of the non-state-space modeling methods to analyze security in terms of attack, detection and mitigation [5]. As to state space modeling, Madan *et al.* applied stochastic reward nets (SRN) to quantitative assessment of security attributes for an intrusion tolerant system [6]. Markov chains have been used to analyze cyber security with attack graphs [7]. The other method is to provide some formal methods to represent or identify vulnerabilities and attacks [8]. In [9], a frame- work considering timed security requirements was proposed by using timed extended finite state machine. A logic-programming approach is presented to analyze network security [10]. They focused on the interactions among different hosts instead of a single host. This method could discover attack paths in different hosts, but cannot identify unknown vulnerabilities or intrusions in a single system. What is more, the above re- searches did not consider source codes and hence did not provide enough information for mitigation strategies. Chen *et al.* [2] explored code-level modeling. They applied data-driven finite state machine to analyze vulnerabilities. In the model, an object, which did not conform to specification at the same time did not have corresponding implementation, will transfer to accept state. However, an object which conformed specification will also transfer to accept state. Therefore, it is difficult for practitionersto understand this model clearly due to mixture of secure and unsecure transitions.

As to analysis of attack model, Andrew *et al.* [11] provided a method to record attack information in a structured and reusable form which includes attack pattern, attack pro-file and attack tree. However, this method only considered the attack patterns dedicatedto buffer overflow and web attacks. Michael *et al.* [12] developed attack patterns to locate vulnerabilities in software design stage in order to reduce the cost of finding andfixing vulnerabilities in maintenance and testing phases. However, the number of attack pattern categories in the attack library was too large to obtain the similarity among exploits. It is difficult, if not impossible, to classify a new vulnerability with existing classification, since this method focused on too many details. What is more, it only described operations in attack path and did not consider vulnerable code. Therefore, it is hard to get efficient countermeasures from the model [12]. Bozic *et al.* [13] presentedan approach to take advantage of attack patterns, modeled by UML state machine, for test case generation and execution. Nonetheless, this method was only applied to XSS attack pattern and it is not clear whether it could be extended to other attack patterns. Kshirsagar *et al.* [14] developed a network application to detect intrusion. Their tool was based on CIDF architecture. Bozic *et al.* combined a combinatorial testing and a model-based technique to generate test cases [15]. However, these researchers focused on web vulnerabilities instead of system vulnerabilities. In fact, not many researchers are involved in attack model analysis for system vulnerabilities.

In this paper, we propose a comprehensive modeling approach to apply attack modelsin the form of code level finite state machines to identify potential vulnerabilities in operation system. This method not only can be used to reason about potential vulnerabilities, but also provide efficient mitigations in design, implementation and testing phases.

3. Problem Statement

There are many kinds of complex reason on VM escape, therefore it is difficult to summarize and extract the characteristics. If we can describe its attack behavior and monitor it keeping in step with the change of system state, it will be helpful to extract and protect the attack model of VM escape.

3.1 Elevated Privilege Model

The attackers take advantage of the virtual machine operating system to initiate requests for execution of sensitive instructions, which are processed by the kernel state, and some privileged instructions are referred to hypervisor for processing. At this point, anattacker can exploit the fragility vulnerability of hypervisor to enable hypervisor to execute a privileged instruction without returning the instruction state, causing the user state to stay in the kernel state, and the attacker implements the privilege elevating, thencan penetrate the hypervisor and other areas of the virtual machine to destroy the isolation mechanism of virtualization, ultimately complete the escape operation.

The following are four conditions required to complete a successful escape attack:

1. A vulnerable kernel;
2. An escape of a matching loophole;
3. Having the ability to transfer the escape to the target position;
4. Having the ability to perform an escaping attack at the target location.

Privilege elevation is particularly important in virtualized systems, including two- level privilege elevation, from the user layer to the kernel layer, and from the kernel layer to the virtualization layer. Elevation of privilege can enable attackers to gain higher privileges, run higher levels of code, and produce greater harm.

The purpose of the permission elevation are three aspects:

1. Reading / writing any sensitive files;
2. The attack is still running after system restarting;

Inserting the permanent back door.

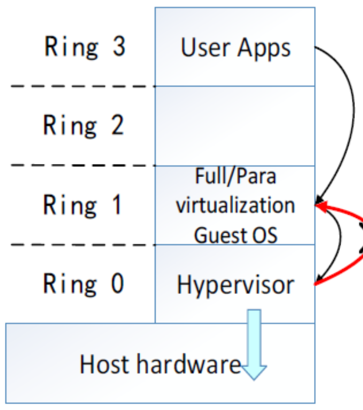

Fig. 1. a. Elevated privilege models of full

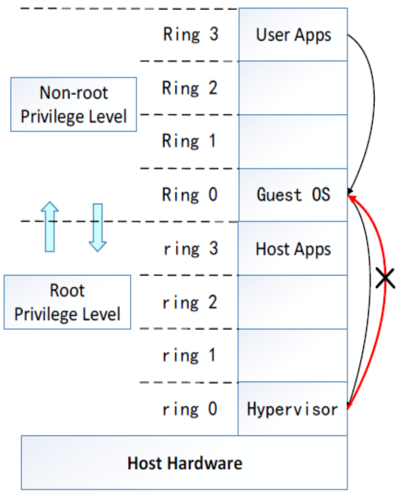

Fig. 1b. Elevated privilege models of hardware-virtualization and para virtualization assisted virtualization.

The elevated privilege models of full virtualization and para virtualization are shown in Figure 1a:

Definition 1: $S = \{S_i\}$ is the collection of all sensitive instructions in the x86 host system, $i \epsilon [0,\infty)$. These sensitive instructions include instructions for accessing or modifying the virtual machine mode or host state, accessing or modifying sensitive registers or storage units, such as clock registers, interrupt registers, accessing the storage protection system and address assignment system, and almost all I/O instructions. The execution of these instructions will change the level of privilege of the original instruction, which may lead to the occurrence of the privilege elevating.

Definition 2: $r = \{r_i\}$ is the collection of privilege levels representing the host operating system, $0 \le i \le 4$, that is $r = \{r_0、r_1、r_2、r_3\}$ .The user application is at r_3 level, the system driver is at r_2 level, the virtualized guest OS is at r_1 level and the hypervisor is at the highest level r_0 .

Definition 3: $S_i <r_j>$ represents a sensitive instruction S_i at a privileged level r_j , $i \epsilon [0, \infty)$, $0 \le j \le 4$ Definition 4: $I = exe(S_i <r_j>)$ represents the execution of a sensitive instruction S_i , and I takes a value of 0 or 1. If $I = 0$, the execution of a sensitive instruction S_i is an escape process, and if $I = 1$, the sensitive instruction S_i is performed correctly by the system.

Definition 5: $r_k = rtn(I) = rtn[exe(S_i <r_j>)]$ represents the final return privilege level of a sensitive instruction S_i . If $r_k = r_j$, where $k = j$, the sensitive instruction S_i returns after normal execution, if $r_k = r_0$,the sensitive instruction S_i is an escape process, and user state application elevated to highest privilege level successfully.

Similarly, the elevated privilege model of hardware-assisted virtualization is shown in Figure 1b, the execution of sensitive instructions requires the jump from a non-root mode to the root mode, and the final escape successful or not is also decided by the results of sensitive instructions to return to the original mode, or stay in ring0.

3.2 Analysis of VMCS Pointer Instruction

Intel and AMD vendors introduced hardware-assisted VT (Virtualization Technology) in 2005, which resolved the reliability and performance deficiencies of pure software virtualization solutions, while virtualization technology is more biased towards full virtualization. Hypervisor can also capture CPU instructions, act as intermediaries for instruction access hardware controllers and peripherals, and there are more virtual ma- chine escape problems based on hypervisor. VT-x expands the operations of the virtualization system to two forms: VMX root operation and VMX Non-root operation. As shown in Figure 2, this paper studies the application of a broader virtualization architecture based on Intel processors.

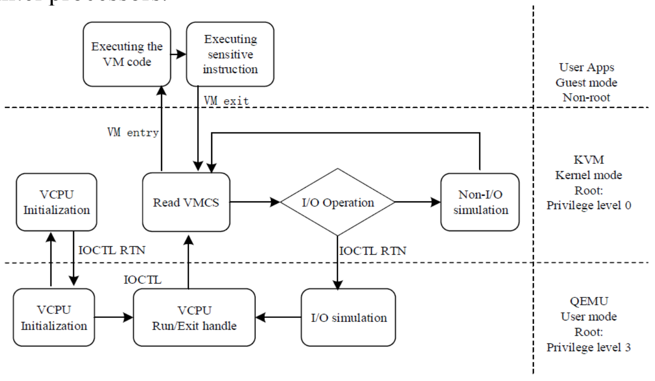

Fig. 2. Instruction processing work model in KVM virtualization system.

The workflow of Figure 2 is:

a) QEMU running in the user state with root mode uses the IOCTL system call to manipulate /dev/kvm character devices, creating VMs and VCPU;
b) The kernel KVM module in root mode is responsible for initializing the relevant data structure, then returning to the user state;
c) QEMU runs VCPU processing through the IOCTL call, that is, scheduling the corresponding VM operation;
d) After the kernel carries out the related processing, it executes the VMLAUNCH instruction, and enters the client operating system to run through VM entry, theclient operating system runs in the non-root mode;
e) The client operating system executes the corresponding virtual machine code, and the insensitive instruction can run directly on the physical CPU;
f) When a sensitive instruction, such as an external interrupt occurs, or an internalexception, occurs on the client operating system, a VM exit is generated and therelevant information is recorded in the VMCS structure;
g) VM exit causes the CPU to return to root mode, read VMCS structure to judgeVM exit by VMM;
h) If I/O or other peripheral instructions, then return to the user state QEMU (thatis, the root mode of ring3), and QEMU completes the simulation of the relevantinstructions;
i) If not, it is handled by the VMM itself;
j) After processing completes, VM Entry enters into the client operating system.

The QEMU thread interacts with the IOCTL between the KVM kernel module and the client operating system, and the KVM kernel module switches with the client soft- ware through VM exit and VM entry operations. The hypervisor and client operating systems share the underlying processor resources, so the hardware requires a physical memory area to automatically save or restore the context in which they are executed. This area is called the virtual machine control structure VMCS, the maximum size doesnot exceed 4KB, each VM needs to correspond its own VMCS, and VMM uses VMCSto configure VM running environment and to control VM running. VT designs a control structure VMCS for each virtual machine and hypervisor to preserve their information. For AMD processors, it is VMCB (virtual machine control block). VMCS is stored in memory and operated by the CPU, holding the information and content of VCPU related registers. That is: a physical CPU can obtain various information of eachvirtual CPU through VMCS. In addition, VT also provides a number of instructions forthe CPU to direct access to VMCS. The system IOCTL file and real-time data collectedfrom the VMCS can represent the current state of the system, and the intrusion detectionsystem can assist to establish the rule base according to the VMCS instruction sequence and compare the vulnerability characteristics according to the rule base.

Intrusion detection system designed combines semantic parsing of IOCTL file and instruction analysis of VMCS pointer instruction in the virtualization system, to collectand describe the original state instruction sequence among the virtual machine, QEMUand hypervisor, then to establish the escape behavior of state sequence feature library,solving the behavior description problem of the virtualization system.

3.3 Feasibility Analysis By FSM

- Root of FSM

State is one of the commonly used and undefined concepts in system science, which refers to the condition, situation and characteristics of the system which can be observed and identified. If correctly distinguishing and describing these states, we grasp the system. When describing a system with a finite state machine, the system should meet different conditions, and can be divided into different states. These finite states will be transferred after the occurrence of a particular event, and the event that causes system state to transfer is also finite. When the system behavior can be divided into a finite state, the system shows the state behavior. Finite state machine is an effective method to stipulate the whole behavior of the system. Being in a state means that the system only responds to a subset of all allowed inputs, produces a subset of possible responses, and that changing state is also only a subset of all possible states.

- Process of VM escape

With the continuous emergence of VM escape events, the intrusion methods are gradually transformed from the initial lower level of single step passive attack into complex and premeditated multi-step attacks. The attack process of the network intruder usually includes several discrete attack steps, each step attack is often an independent attack behavior, and the intruder has a lot of methods to achieve the same goalor the effect of the right operation. Because of the stage characteristics of multi-step attacks, intruders can use a variety of methods to achieve the same target, which makesit more difficult to detect multiple step attacks. At present, the main methods of attack detection are very limited, which can only detect the multi-step attack behavior by matching the pattern of the preset vulnerability database. If only for the single step attack process detection, it is difficult to involve the entire attack process, therefore, thedetection should not be isolated by the individual attack steps, and should be detected by relevant perspective of multiple attack steps.

- Influence of VM escape

Before an attacker carries out an escape attack, the first thing to do is to figure out the method of how to elevate, what kind of goal to achieve. The common methods include destructive and invasive types. A disruptive attack is a denial of service attack that simply destroys the target and makes it work improperly. The intrusion type attacktakes the control target as the core, obtains the more information from the system. Thisattack is more prevalent and more threatening than a destructive attack. Once you get the administrator privileges on the target, you could do anything arbitrarily, including a destructive attack. In order to achieve their goals, attackers firstly take certain measures to obtain information about the target host or the target network, such as usingport scans to obtain port number of the target host system. In order to improve their privileges, attackers often use the target host system vulnerabilities to perform some unconventional operations.

Regardless of the attack method the attacker takes, it will have a certain impact on the target host system when the attack is implemented. For example, in the QEMU simulation process, the right is raised by occupying the host's computing resources through an incorrect call, and if the attacker triggers an interrupt service in the host system, it will cause the buffer to be accessed. If the CPU resource occupied by the incorrect call or the memory resource overflowed is regarded as a system resource, thenthe attacker will change the assigned resource value of the system during the executionof the attack.

From the analysis above, it is known that the escape attack is not only procedural, but also every step in the attack process may change the characteristics of certain re- sources within the attack target itself or the attack target, so that the system passes fromthe secure state to some abnormal middle state and is finally to an insecure state, whichis completely attacked. The system state is divided by the value of system resources, then the system will be in a specific state at any time, and these states are finite. Whenthe system is under attack, the value of system resources will be changed, resulting in the transfer of system state. If changing system resource values is regarded as events that cause state transitions, it is obvious that these events are also finite. According to the nature of the finite state machine, we can completely describe the attack process with a finite state machine. And the finite state machine has a "memory" function be- cause its output is related to the current state. Describing the attack behavior with a finite state machine can reflect the procedural nature of the attack, so as to truly reflectthe escape behavior.

3.4 Description of Elevated Penetration Attack Model

The attack model is an abstraction of attack behavior. Summarizing the attack model must be derived from the actual code level of the attack behavior, but it cannot focus too much on a particular type or a type of code and lead to a general decline. In this paper, the design of virtual machine escape privilege escalation attack model is de- signed to analyze the loopholes of current NVD about KVM and XEN virtualization system.

When describing an escape behavior with a finite state machine, it is necessary to firstly analyze the attack scenario to determine which resources in the system will be affected when the attack is implemented and what kind of hazards will be caused to the system. Then, according to the change of the resource values in the system, divides the system state and determines the attack event that caused the state transition. Finally, a formal finite state machine can be used to describe the attack process.

Assuming that the amount of escape attacks in the system is numbered, the following attack i can be described as follows:

Definition 6: Virtual Machine Escape Attack

$Attack_i = \{Q_i, \Sigma_i, \delta_i, qs_i, T_i, \varepsilon\}$.

1) Q_i is the instruction sequence status set of the i attack, which represents the state of the system at this time. Each element of the instruction sequence represents thepossible state of the target system when the system is under attack. These instruction sequence states are based on the system resources affected by the attack;
2) Σ_i states that the collection of attack events that may cause abnormal changes in system resources during the attack;
3) $\delta_i : Q_i \times \Sigma_i \rightarrow Q_i$,is the reflection from $Q_i \times \Sigma_i$ to Q_i , Since the uncertain finite

state machine is used, the mapping result is a subset of Q_i ;

4) $qs_i \in Q_i$ states the only infinite state of the i attack, which indicates the security status of the target system before the attack;
5) $T_i \subseteq Q_i$ and $T \neq \varnothing$, which indicates the set of final states of the attack, indicating that the attacker has completed the attack or identified the attack on behalf of thedetection system;
6) ε indicates the threshold for the occurrence of this

attack. This value is used for later analysis of the detection system.

Because the escape attack will cause the target system from the secure state to some intermediate abnormal state and eventually become insecure, the state in the Q_i can be divided into three categories, that is $Q_i = \{qs_i, qa^j_i, qd_i \mid j = 0 \text{ or } j = 1, 2, \ldots, M\}$.

a qs_i indicates the security status of the target system before the attack;

b qa^j_i indicates the j security status of the target system after the attack occurs and before it is completed, $j = 0$ indicates no middle state, indicates the number of the ab normal middle state;

c qd_i indicates the system's escape or failure state after the attack is completed. e_i^m indicates the m element of Σ_I, e_i^m is the m attack in the attacking process, $m = 1, 2, \ldots, N$, N is the number of Σ_i . δ_i indicates the state transfer function, which takes the system state and attack events as the input, and takes Q_i non-empty subsets as the output, such as $\delta_\iota(qa_i^j, e_i^m) \rightarrow \{qa_i^j, qa_i^m\}$, $qa_i^0 = qs_i$ while $j = 0$, $qa_i^M = qd_i$ while $m = M$; qd_i indicates the unique end state of the attack, and $Ti = \{sd_i\}$.

4. Proposed the Modeling Language and BasicModel

This section firstly introduces a formal language, an extension of Datalog is defined, that includes clauses (operation and judgment), variables and constants, then raises thebasic elevated penetration attack model.

4.1 Datalog Overview

Datalog, as a data query language on logic-based, inspired by the Prolog (Programming in logic) language, is subset of Prolog. Its sentences are made up of facts and rules as the same as Prolog, which can implement deductive reasoning for a knowledge base, that is, a new fact could be obtained from the known facts by inference. A rule of Datalog consists of the following three parts:

1) Head: A relation atom.
2) Implication symbol: ":-", read as if, indicates a logical relation rather than an operational symbol.
3) Body: Include one or more atoms. Atoms may be related or arithmetical, with constants or variables as parameters. Each atom is connected by AND, and the atom with NOT ahead is called the counter atom.

A sentence can be represented as a clause:

$$P :- P_1, P_2, \qquad , P_n$$

This clause means that if $P_1, P_2, \ldots\ldots, P_n$ are true, then P is true. Otherwise, P is false. The left part of the clause is called head, while the right part is called body. If a clausehas non-empty body, it is called a rule. If a clause has no body, it is called fact.

4.2 Datalog-based Formal Language

Datalog has limited forms of clauses. This section presents a modeling language which allows a generic and succinct representation of exploitation. The formal language con- sists of a clause, variables and constants, defined in TABLE 1, 2 and TABLE 3. The clause is further divided into two different types: action and judgment. The vulnerability reports provide information about vulnerable reason and exploits. The language must define literal, variable and constant. The description is abstracted into literals, which is divided into two different types: operation and judgment. Operation is operation conducted by attackers. Judgment is a check about whether some conditions con- form to our designed security specification and how much probability matching escape vulnerability.

Table 1. Operation and Description

Literal	Description
ChngPt(principal, orgPt, newPt)	A principle changes original pointer to new pointer.
GetLength(p)	Finding the length of object P.
WriteBuf(data)	Writing data with size into buffer.
ExeOpers(oprA, oprB)	Executing operation A, then executing operation B.
ChngRTN(p, data)	Modifying the return value of p and writing data.
EoP(OperA,formerPriv, latterPriv)	Elevating privilege of operation A from former-Priv to latterPriv.
CallFunc(func)	Calling function Func.
ExeCode(principal, program, priv)	A principle executes program with privilege.
Access(principal, r/w, data)	A principle accesses (read/write) data.
Dos(principal, program, priv)	Generating denial of service under privilege.
Oob(principle, operA)	A principle conduct out of bound operation A (access/write/read).

Table 2. Judgment and Description..

Literal	Description
NotPositive(Var)	Checking whether Var is non-positive.
Sign(Var)	Checking whether Var is signed integer.
Uninitialized(Var)	Checking whether Var is uninitialized.
Uncleared(Var)	Checking whether the value or information Var is cleared.
Free(Var)	Checking whether Var is freed.

Table 3. Description of Variable and Constant.

Variable	Description
orgPt	Original pointer
newPt	New pointer
RTN	Return address
oprA	Operation A
oprB	Operation B
formerPriv	The privilege before certain operation
latterPriv	The privilege after certain operation
func	Function
Var	Variable
priv	Privilege

As shown in Fig. 3, the basic elevated penetration attack model based on finite state machine includes six states [16]: start state (S), specification check state (SC), weak secure state (WS), insecure state (IS), good state (G), failure state (F). In view of the definition 6, S represents qs_i and $qa_i{}^j$ involves SC, WS and IS, then qd_j is equal to F. It also includes transitions: Implementation conforms to security specification completely, Implementation does not conform to security specification completely, deep checking in instruction sequence how much probability matching escape vulnerability.

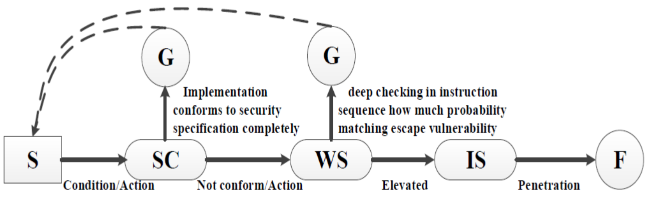
G
G
Implementation conforms to security specification completely
deep checking in instruction sequence how much probability matching escape vulnerability
S
SC
WS
IS
F
Condition/Action
Not conform/Action
Elevated
Penetration

Fig. 2. Basic elevated penetration attack model based on FSM.

In Fig. 3, there is a label "Condition/Action" attached to every transition. Condition means the pre-condition to trigger the transition. Action is the post-condition caused by transition. From S, the model will check whether condition is satisfied. If yes, it goes to check. The security specification describes system secure behavior. If so, the condition of the object will transfer to good state (G). If the code is not in accord with security specification completely, the model will go to weak secure state (WS). Then the model will check whether there is a deep check in instruction sequence how much probability matching escape vulnerability and how it works in implementation. If the probability δ is less than the set threshold □ , meaning not a VM escape vulnerability, the model will transfer to good state (G). Otherwise, the model will transfer from WS to IS, indicating an elevated operation to be the successful intrusion. Therefore, apart from the lack of security mechanism completely, this model could also describe in-sufficient security assurance mechanism. At last, the condition will transfer from IS to F along with penetration by executing arbitrary code, causing denial of service, or accessing arbitrary data. Hence a VM escape behavior is completed entirely. The basic elevated penetration attack model is used to describe every step of attack pattern. Consequently, we can acquire FSM model of attack patterns by combining every basic FSM [2]. In this section, our models describe escape process conceptually with condition and action represented by Datalog based language in FSM.

5. Proposed the Different Attack Models

Most of VM escape vulnerabilities could be exploited by certain different methods. According to NVD security research, we generalize four attack models that include specification violation attack, buffer overflow attack, out-of-bounds attack and error handling attack. This section presents FSM models of the above attack patterns. Each model consists of one or several basic elevated penetration attack models described inSection 4.3. These models offer general steps to conduct escape conceptually and per-form as the templates for every attack pattern which is helpful to analyze specific casesin vulnerability database. The pre- and post-conditions in FSM are represented by the formal language presented in Section 4.2. According to attack models, mitigation strategy could also be gained.

5.1 Specification Violation Attack Model

Specification violation attack model often works at the judgement stage of SC. The representative attack patterns of this model are type confusion attack, uninitialized at- tack, use after free attack, and so on.

The general steps of specification violation attack model are shown in Fig. 4. In the SC, FSM will check whether implementation accords with security specification completely (e.g. whether type of argument matches predefined parameter type, whether un- initialized variable/object with previous information remained in the memory, or whether certain vulnerable operations leading to freeing of variables but not clearing the reference to the variables). If so, the system is secure. Therefore, the control flow will exit the model. Otherwise, it transits from SC to WS and the attacker has passed the first check. At state WS, the model will check whether there is a deep check in instruction sequence how much probability matching escape vulnerability. If the prob- ability δ is less than the set threshold ε , meaning not a VM escape vulnerability, the model will transfer to good state (G) and the control flow will exit the model. Other- wise, it will turn to state IS. At IS, the attackers craft variable type to manipulate addressspace to elevate (e.g. Make pointer of certain class method point to malicious shell code) and have passed all of checks. The attackers can execute the program with privileges of owner.

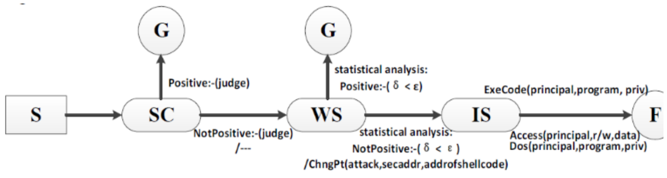

Fig. 3. Specification violation attack model based on FSM.

In Fig. 4, following the attack path, specification violation attack model and corresponding mitigation strategy could be derived. The attack model is described by Data-log based language. According to Section 4, the sentence below means in order to exe- cute program with owner's privilege for attacker, the steps below should be followed. Firstly, the buffer overflow vulnerabilities should exist. Then ensure that implementation doesn't accord with security specification completely. At last, the attackers have already changed function pointer successfully.
Take an example of use after free attack, attack model description:
ExeCode(Attacker, Program, Owner):-
GetVariable(PtToVar);
isUncleared(PtToVar);
Not isNtFree(Var);
AllocMem(Length, PtToObject);
ChngPt(Attacker, PtObjectMethod, AddrOfShellCode);
isMalicious(Attacker).
Mitigation:
In order to prevent UAF attack, there are three countermeasures:

1) The mechanism to check whether pointer to the variable is freed;
2) The mechanism to check whether the variable is freed;
3) The mechanism to check whether pointer of object method is modified.

5.2 Buffer Overflow Attack Model

Buffer overflow attacks mainly include integer variable overflow, stack smashing, heap corruption and so on. The general steps for buffer overflow attack model are shown in Fig. 5. The attacker allocates destination buffer determined by the given length (some- times this is predefined by program). Then attacker writes data to buffer, goes across the boundary of buffer and overwrites memory locations (e.g. Make return address point to malicious code). Shellcode will be executed when function returns. Ellipsis represents accumulation process, which means that it will take a long time to input data into buffer and overflow it. The attack model has multi-WS process used to check whether the length of destination buffer is non-negative, after this check, other potential vulnerabilities related to negative buffer size can be ruled out. The description of the attack model in the formal language and attack mitigation strategy are derived below.

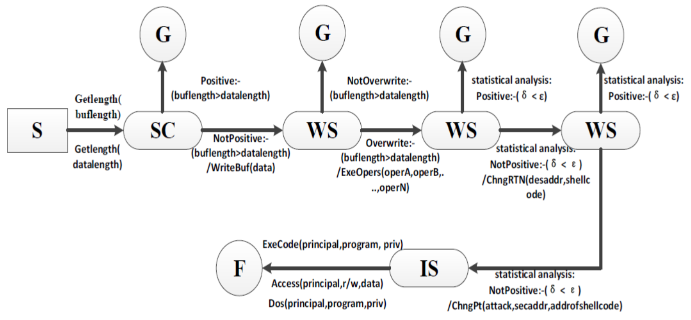

Fig. 4. Buffer overflow attack model based on FSM.

Attack model description:
ExeCode(Attacker, Prog, Owner):-
GetVariable(BufLength);
AllocMem(BufLength, DesBuf);
isLargerThan(InputSz, maxBufLength);
isLargerThan(ε , δ);
ChngPt(Attacker, RTN, addrOfShellCode);
isMalicious(Attacker).
Mitigation:
In order to prevent buffer overflow attack, there are two countermeasures:

1. The mechanism to check whether input size is within buffer length;
2. The mechanism to check whether return address (RTN) is changed.

5.3 Out-of-Bounds Attack Model

The general steps of out-of-bounds attack model are shown in Fig. 6. Out-of-bounds attack model, pointer index failure leads to be intended for use, always causes denial ofservice of the system. It often appear in self modified KVM or XEN virtualization systems. It is possible that request data length will go beyond the array available length. Then control flow pointers (e.g. function pointer, return address) can be modified to point to malicious code. Then the malicious code will be executed when control flow transfers to modified pointers. According to Fig. 6, the description of out-of-bounds attack model in the formal language and the attack mitigation strategy can be derived.

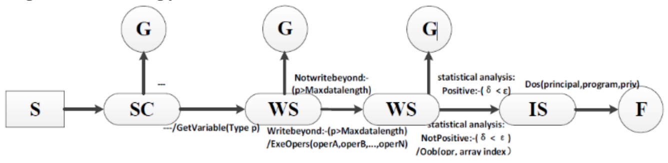

Fig. 5. Out-of-Bounds attack model based on FSM.

Attack model description:

ReqCheck(CndEqtn):-
GetVariable(Type p);
isLargerThan(p, Maxdatalength);
isUsedBy(p, equation or array index);
isLargerThan(□ , δ);
isOob(equation or array index);
isMalicious(Attacker).

Mitigation:

In order to prevent out-of-bounds attack, we can apply two mechanisms:

1. The mechanism to check whether request data length is beyond the array availablelength;
2. The mechanism to check whether pointer index is failure.

5.4 Error Handling Attack Model

The general steps for error handling attack are shown in Fig. 7. Firstly, the attackers tryto call a function to trigger an interrupt. Then they will confuse interrupt processing mechanism so as to stay ring0. At last, control flow pointer (e.g. function pointer, returnpointer) is overwritten to execute arbitrary code. According to Fig. 7, the description ofmodel and the attack mitigation strategy can be obtained.

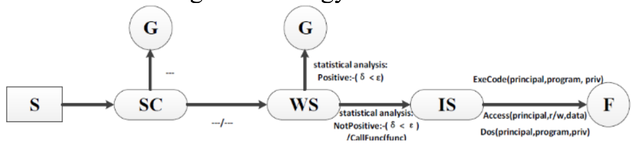

Fig. 6. Error handling attack model based on FSM.

Attack model description:
isLargerThan(□ , δ) :-
CallFunc(func);
ExeCode(principal, program, priv);
Access(principal, r/w, data);
isMalicious(Attacker).
Mitigation:
In order to prevent error handling attack, we can apply a mechanisms: the mechanism to check whether interrupt handling is working.

The conceptual models can be used to help identify potential VM escape vulnerabilities and guide the practitioners, especially system engineers with little knowledge of security, to improve system security by adding checking mechanisms in design stage. They can also be detailed by inspecting the vulnerable code after the implementation stage which will be introduced in section 6.

6. Statistics and Evaluation

This section first briefly introduces NVD, an open vulnerability database, then the results of VM escape attack model classification are presented. At last, in order to verify the validity of the attack model, our modeling methods are exemplified by analyzing a recent specific vulnerabilities reported in NVD.

6.1 Statistical Analysis

Virtualization vulnerability source

Our analysis is based on approved vulnerability database, National Vulnerability Data-base. There are 432 vulnerability reports involving 111 KVM virtualization vulnerabilities and 321 XEN virtualization vulnerabilities, which are reviewed until April, 2020.This database not only has validated the majority of vulnerabilities, but also provides more detailed information for every vulnerability, such as vulnerability report date, vulnerability fix date, vendor, patch, and ways to exploit vulnerability. What is more, NVDincludes databases of security checklist references, security-related system flaws, mis- configurations, product names, and impact metrics.

VM escape attack model classification results

We distill 39 VM escape vulnerabilities according to VM escape definition from 111 KVM virtualization vulnerabilities, while 100 VM escape vulnerabilities from 321 XEN virtualization vulnerabilities. Table 4 and table 5 summarize the number and per-centage of each attack model virtualization from different virtualization systems.

As table 4 and table 5 show, buffer overflow attack model accounts for a greater proportion. 2 (5.1%) vulnerabilities in KVM reports and 6 (6%) vulnerabilities in XENreports are related to specific defects in hardware design. The exploit to these vulnerabilities needs to take advantage of specific hardware features. Therefore, there are no general attack models which can cover them. Specification violation attack model focuses on the judgement of SC. If the checking does not conform to security specification completely, it will turn to butter overflow to a large extent. So attack models are much more related to manipulation of address space which is still the main vehicle forattackers to intrude system. Efficient techniques to detect anomalies in address space can contribute so much to preventing intrusion.

Table 4. Number and Percentage of Each Attack Model on KVM Virtualization.

Attack model	Number (%)
Specification Violation Attack Model	9(23.1%)
Buffer Overflow Attack Model	11(28.2%)
Out-of-Bounds Attack Model	9(23.1%)
Error Handling Attack Model	8(20.5%)
Hardware	2(5.1%)
Total Number	39

Table 5. Number and Percentage of Each Attack Model on XEN Virtualization.

Attack model	Number (%)
Specification Violation Attack Model	21(21%)
Buffer Overflow Attack Model	35(35%)
Out-of-Bounds Attack Model	12(12%)
Error Handling Attack Model	26(26%)

Hardware	6(6%)
Total Number	100

6.2 Attack Case Evaluation

In this part, our modeling approach is exemplified by analyzing several specific vulnerabilities reported in NVD. The completion of an escape behavior is often the resultof the interaction of one or more attack models, only buffer overflow attack model combined with Error handling attack model will be detailed due to space limitation, like CVE-2016-3960, CVE-2012-0217, is triggered by SYSCALL and Hypercall instructions, eventually triggering the GP (General Protection) fault with ring0 privilegesand attacker-controlled registers when executing the SYSRET instruction. The model is derived shown in Fig. 8. In this case, the mechanism to check whether buffer lengthis positive should be conducted to avoid other potential vulnerabilities. Then the at- tacker keeps writing data into buffer and our model will check whether the size of inputis within buffer length. If size of input is larger, it will transit to WS and go to second WS. Otherwise, it will go to G. Then the attacker will overwrite memory. In the third WS, the model will check if the RTN is changed by attacker to point to shellcode. If so,the model will go to IS. Thus, the attacker will execute the program with owner's privilege. The attack scenario and mitigation can be obtained as below. The attack pattern is described by Datalog based language.

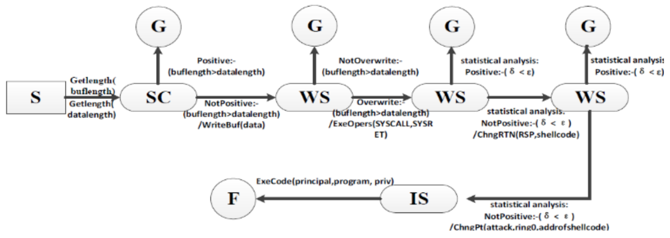
G
G
G
G
S
Getlength(buflength)
Getlength(datalength)
SC
Positive:-(buflength>datalength)
NotPositive:-(buflength>datalength)/WriteBuf(data)
WS
NotOverwrite:-(buflength>datalength)
Overwrite:-(buflength>datalength)/ExeOpers(SYSCALL,SYSRET)
WS
statistical analysis: Positive:-(δ < ε)
statistical analysis: NotPositive:-(δ < ε)/ChngRTN(RSP,shellcode)
WS
statistical analysis: Positive:-(δ < ε)
statistical analysis: NotPositive:-(δ < ε)/ChngPt(attack,ring0,addrofshellcode)
IS
ExeCode(principal,program, priv)
F

Fig. 7. Vulnerability details of buffer overflow attack model combined with Error handling at-tack model based on FSM.

Attack model description:

ExeCode(principal,program, priv):-
Getlength(buflength);
Getlength(datalength);
WriteBuf(data); ExeOpers(S
YSCALL, SYSRET);
isLargerThan(□ , δ);
ChngRTN(RSP,shellcode);
isLargerThan(□ , δ);
ChngPt(attack,ring0,addrofshellcode).
isMalicious(Attacker).

Mitigation:

In order to prevent buffer overflow attack combined with Error handling attack, there are three countermeasures:

1. The mechanism to check whether input size controlled is larger than buffer length;
2. The mechanism to check whether return address (RTN) is changed;
3. As to SC, the mechanism to check buffer length is positive should be conducted to avoid other potential vulnerabilities.

7. Conclusion

In this paper, we innovatively propose an elevated penetration attack model based on FSM, extending Datalog language to describe pre- and post-conditions of exploits for VM escape vulnerabilities specifically, eventually details the escape vulnerabilities, summarizes the key steps, and deduces the escape vulnerabilities. The evaluation experimental results demonstrate that the proposed attack models are reasonable and ac- curate, making up for the evaluation on the VM escape behavior in the virtualization security area. In the future, we plan to apply the elevated penetration attack models to identify more potential VM escape vulnerabilities combined with the virtualization technology and more details of attack behavior. Furthermore, we also intend to implement our approach to different versions of virtualization systems to generalize findingsand refinement of the work.

References

[1] Jenni Susan Reuben. A survey on virtual machine security. On TKK (Helsinki University of Technology) T-110.5290 Seminar on Network Security, 2007: 1 (8).

[2] Shuo Chen, Jun Xu, Zbigniew Kalbarczyk, and K. Iyer. Security vulnerabilities: From analysis to detection and masking techniques. Proceedings of the IEEE, 2006: 94(2):407–418.

[3] Walaa Eldin Moustafa,Vicky Papavasileiou,Ken Yocum,Alin Deutsch.Datalography. Scaling datalog graph analytics on graph processing systems [M]. 2016 IEEE International Confer- ence on Big Data (Big Data), 2016: 56-65.

[4] Arpan Roy, Dong Seong Kim, Kishor S. Trivedi. Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees. DSN 2012: 1-12.

[5] Arpan Roy, Dong Seong Kim, Kishor S. Trivedi. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. Security and Communication Networks. 2012: 5(8): 929-943.

[6] Bharat B. Madan, Manoj Banik, Bo Chen Wu, Doina Bein. Intrusion Tolerant Multi-cloud Storage. 2016 IEEE International Conference on Smart Cloud (SmartCloud). 2016:262-268.

[7] Subil Abraham and Suku Nair. Cyber Security Analytics: A stochastic model for Security Quantification using Absorbing Markov Chains. 5th International Conference on Networking and Information Technology, ICNIT. 2014.

[8] Mohit Dua, Himanshi Singh. Detection & prevention of website vulnerabilities: Current scenario and future trends. 2017 2nd International Conference on Communication and Electronics Systems (ICCES). 2017: 429-435.

[9] Wissam Mallouli, Amel Mammar. Ana R. Cavalli: Modeling System Security Rules with Time Constraints Using Timed Extended Finite State Machines. DS-RT 2008: 173-180.

[10] 1Ou, Xinming, Sudhakar Govindavajhala. and Andrew W. Appel. "MulVAL: A Logic-based Network Security Analyzer." USENIX security. 2005.

[11] Moore, Andrew P., Robert J. Ellison, and Richard C. Linger. Attack modeling for information security and survivability. No. CMU-SEI-2001-TN-001. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2001.

[12] Michael Gegick, Laurie Williams. Matching attack patterns to security vulnerabilities in soft- ware-intensive system designs[C]//ACM SIGSOFT Software Engineering Notes. ACM, 2005, 30(4): 1-7.

[13] Bozic, Josip, and Franz Wotawa. "XSS pattern for attack modeling in testing. "Proceedings of the 8th International Workshop on Automation of Software Test. IEEE Press, 2013.

[14] Kshirsagar D D, Tagad D K, Sale S S, et al. Network Intrusion Detection based on attack pattern[C]//Electronics Computer Technology (ICECT), 2011 3rd International Conference on. IEEE, 2011, 5: 283-286.

[15] Bozic J, Garn B, Kapsalis I, et al. Attack pattern-based combinatorial testing with constraints for web security testing[C]//Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on. IEEE, 2015: 207-212.

[16] Kamil Mielcarek, Alexander Barkalov, Larisa Titarenko. Designing Moore FSM with un- standard representation of state codes [M].2016 5th International Conference on Modern Cir- cuits and Systems Technologies (MOCAST), 2016:1-4.