VIRTUAL MACHINES DETECTION METHODS USING IP TIMESTAMPS PATTERN CHARACTERISTIC

M.Noorafiza, H.Maeda, T.Kinoshita and R.Uda

Graduate School of Computer Science, Tokyo University of Technology, 1404-1 Katakuramachi, Hachioji Tokyo, Japan

ABSTRACT

Virtual machines (VMs) are underlying technologies of IT solutions such as cloud computing. VMs provide ease of use through their on-demand characteristics and provide huge benefits in terms of lowering costs and improving scalability. VMs are also being used as malware detection systems, and with the rapidly expanding usage of mobile devices, besides of their usage as honeypots, VMs are coming to be used as emulators for detecting malware in apps. This is due to the limited resources, such as processing power, available in mobile devices. Currently, the security of applications for mobile devices is checked by running them in VM environments before they are released to the end user. We argue that such a process may cause or overlook serious security threats to the end user. In particular, if a piece of malware can detect its current running environment, it may change its behavior such that it doesn't perform malicious operations in environments it suspects to be emulators. In this way, when the malware detects that its running environment is on a VM, it may be able to hide from the security system on the VM. This is a potential security hazard for end users, especially users of mobile devices. In this paper, we present a VM detection method that we argue could be used for remotely detecting VM environments. The detection method works by analyzing the pattern of IP timestamps in replies sent from the target environment. The method does not require any installation of software on the target machine which further increase its potential harm if it were to be used by malware to detect VM environments. In this paper, we also present a technique to disguise a real PC machine such that it shows the similar IP timestamp patterns as the VM. By using this technique, malware may not be able to differentiate between a real machine and a VM, thus providing protection to PC end users.

KEYWORDS

Virtual machine, remote detection, security, malware, mobile devices, smartphones.

1. INTRODUCTION

Virtual machines (VMs) are underlying technologies in the information technology industry. Besides their cost effective and scalability benefits that are being utilized in cloud computing technology, VMs are also widely used as security systems for malware detection such as honeypots. As the mobile device market for smartphones, etc., has grown dramatically [1], VMs are also being used as emulators for detecting malware in mobile device applications.

Mobile devices are now able to perform many of the operations that had been exclusively done on PCs. Mobile devices use the same architecture as traditional computers; thus they have the same vulnerabilities and security issues faced by PCs [2]. However, smartphones are constrained by their limited resources, i.e., processing power, battery power, and lack of storage, which prevent the integration of advanced security monitoring solutions that work with traditional PCs. With the integration of mobile devices and cloud computing technology, the lack of resources available to mobile devices for security monitoring solutions could be addressed by delegating the security monitoring and malware detection to VMs in cloud computing facilities [3, 4].

However, despite the attractiveness of this idea, we argue that a malware detection security system using VM may have a critical vulnerability. That is, the malware may try to first detect the environment in which it will be running. Through such a VM detection, malware creators may write programs that will not perform harmful operations such as botnet attacks upon detecting a VM environment, thus reducing the risk for their behavior from being studied and revealed. This would also have serious consequences for mobile device users as any applications that have passed a malware detection system on the VM are considered safe and may gain the user's trust.

The issue of VM detection has been widely discussed by researchers [5-7]. There are a number of techniques for detecting the existence of a VM [8]. However, detection method that is done once the detection program is installed and executed on a host is considered the last method that will be used. This is because if the program or software is installed in a host, its existence might be detected and a signature will be generated that may result in their existence being revealed.

In this paper, we present a method for detecting VM environments by remotely detect a VM without installing any program on the target machine. It works by analyzing the pattern of IP timestamps in replies sent from the target machine to determine whether the target machine is a real machine or a VM.

In the experiments, we sent request packets with the IP timestamp option enabled from the test client to the target machines. By enabling the IP timestamp option in the request packets, the replied packets from the target machines included timestamps showing when the replied packets were generated in the target machines. For comparison reasons, VMs and a real machine were used as the target machine. The VMs were popular virtualization products for open platforms [9], i.e., VMWare [10], Oracle VirtualBox [11] and Xen [12], were used as the VM target machine's hypervisors, and the real machine ran open source Ubuntu Linux as the native operating system.

Our findings show that by analyzing the pattern of the IP timestamps in the replies from the target machines, we were able to distinguish between the VM and real machine environments. That is, our experiments results proved that there is a characteristic IP timestamp pattern in the replied packets that could be used for detecting a VM environment. This is because VMs work by time-sharing host physical hardware and it is impossible for a VM to duplicate the timing activity of a physical real machine even if the VM uses several techniques to minimize and conceal differences in timing performance [13].

As a countermeasure to this VM remote detection method, we propose to disguise a real machine in the same environment as a VM so that it returns the same IP timestamp patterns as the VM. Here, we propose usage of mathematical formula for editing the real machine's system to create delays in timestamps reply packets. Through this formula, IP timestamp pattern from the real machines becomes similar as the ones from VM environment. This countermeasure can provide protection to real machines as malware or attackers can not distinguish the difference between a VM or real machine environment.

2. RESEARCH BACKGROUND

In parallel with the growth of mobile users, there has been a significant increase in malware aimed at gathering personal information from mobile devices as well as PCs. This information could later be used by the malware owner or other third party for their personal profit such as for marketing and selling services on the web or profiteering from online banking information [14],[15]. This growing threat points out the need for users to protect their mobile devices by using anti-virus or anti-malware applications [16]. Furthermore, with the advent of metamorphic and polymorphic types of malware, which have the capability to change their code to avoid detection, have made it harder to protect against malware. Some types of metamorphic malware change themselves every time they infect a new computer, whereas the operation codes in polymorphic malware are encrypted.

Malware detection systems could provide a line of defense for PCs and mobile devices from infection or attack. There are three main methods of detecting malware [17]. The first is called pattern matching. This method compares a prospective target malware with its signature, which is a characteristic pattern in binary. However, with the ever increasing number of targets that needs to be compared, malware detection using this method proves to be very challenging. The second method is a generic one that monitors file activities and processes in computers and prevents certain modifications from being done to the operating system or related files. Operation rules are written in a definition file and antivirus software compares the next operation of a target with the rules. When the next operation determined to be out of the bounds of the rules, it is cancelled. The third method is a heuristic one that is applied before applications are operated. Programs are analyzed by anti-virus software to determine if any operations are performed differently in comparison to when they were executed somewhere else.

The trouble is that anti-virus or anti-malware schemes such as those above are hard to implement on mobile devices that have limited CPU, power and memory resources [18]. In order to conserve mobile resources while improving detection of malware threats, an off-device in-cloud network service could be implemented [19]. Through this approach, security services on VMs in the cloud system scan for malware affecting mobile device applications in order to free up on-device CPU and memory resources of the mobile devices while conferring a high level of malware protection.

VMs are commonly used as honeypots or emulators for malware detection. VMs are used since their environments can easily be restored upon infection by malware. The VMs are implemented on hypervisor hosts. There are two types of VM hypervisor. Type 1 hypervisors, or bare-metal implementations, run directly on the server hardware without any operating systems beneath them, whereas Type 2 hypervisors run on top a traditional operating system. Type 2 hypervisors are easy to install and deploy because much of the hardware configuration work such as networking and storage is handled by the underlying operating system [20]. A VM employs a dynamic heuristic method for scanning and detecting malware. If any of the scanned programs is detected as malware, it could easily be isolated and analyzed in the VM environment with no or little possibility for it to cause any harm on the VM hypervisor or host machine.

However, even with the significant merit of using VMs as a defense against malware, the idea is still vulnerable due to the possibility of the malware to detecting the system on which it is or will be operating and thereby distinguishing the VM environment. Through VM detection, attackers could design malware that first try to detect whether the system is running on a VM or not before executing any malicious or security breaching operations. Moreover, once that point is reached, the attacks can escalate from just VM detection to the exploitation of the VM itself [6, 19]. We also predict that tests of the security of application software for mobile devices will be done using emulators in VMs in cloud computing environments. We posit that the results of application tests might not be true, especially security tests aimed against various malicious code because the malicious operation may not show their true nature if they have already detected that the running environment is a VM. As a result when the application is released, mobile devices that install it might be compromised.

This creates a critical vulnerability since malware that has avoided detection in the VM may be downloaded to end user PCs and mobile devices as trusted applications. Some well-known attacks against VM environments have exploited VM detection methods; they include BLUEPILL [21], DKSM [22] and Subvir [23]. One of the effects of VM detection is that intrusion detection systems on VMs such as Livewire [19] might not show real results. For example, ReVirt [24] uses the hypervisor layer to analyze damage that hackers do during an attack. It might not be able to provide an accurate analysis or fully update its malware behavior signature database.

We argue that intensive studies into VM detection methods are required in order to provide strong protection against malware which includes end users using PCs and mobile devices. However, VM implementations range from those on known to those on unknown hardware configurations on various platforms, and hypervisors and VM detection spans a spectrum of scenarios that need to be investigated. In particular, our remote VM detection method can detect VM environments of type 1 and type 2 hypervisors in the test bed environment that was setup in our laboratory.

Timestamps is 32 bits of milliseconds since midnight UT that are used in various network protocols, such as IP, ICMP and TCP. IP was designed to support extensibility using IP options [25]. IP timestamp options are variable-length data that are stored in the IP header and are associated with a particular extension type. One of the options allows the sender to request timestamp values from any target machine which handles the packet by specifying its IP address. A timestamp is the current time of an event that is recorded by a computer. This research hypothesized that a VM environment could be detected by comparing the behavior patterns of IP timestamps sent from VM target hosts and with the IP timestamps of physical machines within the same cloud computing environment.

Each VM can be configured with one or more network interface cards (NIC), as shown in Figure 1. Hypervisor supports the creation of a virtual network that connects the virtual NICs to a network that is composed of virtual switches. These virtual networks connect to the physical NICs. Networking allows applications on VMs to connect to services outside the hosts. As with other resources, the hypervisor is the manager of network traffic in and out of each VM and the host. Applications send network requests to the guest operating system which passes the request through the virtual switch. The hypervisor then takes the request from the network emulator and sends it through the physical NIC card out into the network. When the response arrives, it follows the reverse path back to the application. As a result, virtualization adds a number of wrinkles to the networking environment. Because of that, real machines and VMs will show differences between the IP timestamps in the replied packets. We predicted that differences would be smaller in real machines compared with VMs because switching between operating VMs in the queue causes the VMs to have a comparatively slower IP timestamp reply pattern. In addition, the VM clock is managed using a timer device emulation called VM switch. This should result in significant differences in the timestamps between reply packets from a VM.

In our research, we exploited the behavior of the VM's clock management through a network implementation of IP timestamp replies characteristic of the VM in order to remotely detect the existence of the VM itself.

3. RELATED WORK

Detection methods that focus on the network implementation and behavior of VM could be considered ways of remotely detecting VMs without compromising the target. A VM detection method that uses network timestamps was first suggested by Kohno [26], wherein the TCP timestamp was used as a covert channel to reveal the target host's physical clock skew. Meanwhile in [27], discrepancies between two different kinds of timestamp, ICMP and IP, in one packet were used to determine the presence of a VM. However with the rapid improvement in high-performance PCs and high-speed network connections, the scenario has become inapplicable. In one of our previous papers [28], we showed that for type 2 VM hypervisors, the replied IP timestamp information received by the client test machine indicate different behaviors compared the IP timestamps from a real machine. In [29], we proved that the IP timestamp patterns for the type 1 hypervisor also show distinguishable differences between real machines and VM. We proved that the IP timestamp differences in the packets from the VM hosts are bigger than those from a real machine because the operating VM will switch with other VM in the queue and the VM clocks are managed by a timer device emulation (called VM switch) , wherein VM operations sometimes pause in order for the other guest operations to be completed. In the current paper, we strengthen our previous findings to show that remote VM detection using IP timestamp patterns can be used for detecting VMs on major hypervisor products.

4. Proposed Remote Detection Method and Countermeasure

Our remote VM detection method works by exploiting the IP timestamp information in the reply packets from the VM. By analyzing the pattern of the IP timestamps, it can reveal the presence of a VM. The IP timestamp is an optional extension to the IP header that allows the sender to request timestamp values from any machine that handles the packet by specifying its IP address [30]. The IP timestamp option has three modes available:

1. Collect timestamps from each device; space in the header is available for up to nine devices.
2. Collect the IP address and up to four timestamps from each device .
3. Specify in advance up to four IP addresses from which a timestamp is requested.

Here, we chose to implement the third mode, as it lets us specify in advance the target host IP address in the controlled environment of the experiments. Another reason for selecting this option is that it is conveniently available in the Linux ping command.

In [27], it was observed that the IP and ICMP timestamp patterns could be differentiated by analyzing the IP and ICMP timestamps in 1,000,000 packets sent back as replies from the target VM. Thus, it is concluded that 1,000,000 IP timestamps from the target machines would be sufficient for observing the differences in IP timestamps patterns between the target machines.

In our client test, we sent 1,000,000 consecutive non-suspicious packets that had the IP timestamp option enabled to the target machines. Both type 1 and type 2 hypervisors of the most popular VM technologies were setup in our laboratory. To avoid the request queue from being flooded, we sent the next request packets after the previous packet reply was received. The reply packets from the target machines included the IP timestamp information, i.e., the time in the target machine when the reply packet was generated. The data structure of the sent IP packet is shown in Figure 2, while the structure of the IP timestamp option packets is shown in Figure 3. The reply packets from the target clients and their IP timestamps were analyzed to determine if there was any distinguishable pattern characteristic between the target machines. In the same way as shown in [28, 29], we could distinguish the IP timestamp patterns of the VMs from those of the real machine. We will discuss these pattern differences in the Data Analysis section.

We plotted distribution graphs on the basis of the timestamp pattern differences identified in the experiment. Then, we devised a solution using our delay modification technique, wherein real machines are made to show the same IP timestamp reply pattern as the VM, thereby eliminating any chance differentiating between VMs and real machines within the network by using IP timestamp patterns. This countermeasure should be implemented in the real machine rather than the VM mainly because VMs work by time-sharing host physical hardware and it is impossible for a VM to duplicate the timing activity of a physical machine even if it uses several techniques to minimize and conceal differences in timing performance [13]. We implemented the modification in the time-stamping process in a real machine to create delays in the timestamps in the reply packets. When the packets with the IP timestamp option arrive at the real machine, they are delayed using the countermeasure before being forwarded to OS for processing. The delay is implemented by adjusting the mean number of repetitions of the same IP timestamp in the real machine to match those in the VMs. The following notation will be used to describe our method to calculate the mean number of repetitions of the same IP timestamp of the real machine and the VMs.

r_i: number of identical timestamps in the i-th run of a real machine
(= i-th run length of identical timestamps)
r_j: number of identical timestamps in the j-th run of a virtual machine
(= j-th run length of identical timestamps)
n_r, n_v : number of runs of repetitions of the same timestamp for a real machine and virtual machine, respectively
R_r, R_v : mean run length for a real machine and virtual machine, respectively
t_r, t_v : mean interval time between successive packets from a real machine and virtual machine, respectively.

The mean run lengths, R_r and R_v, are calculated formulas follows.

$$R_r = \frac{\sum_{i=1}^{n_r} r_i}{n_r}, \quad R_v = \frac{\sum_{j=1}^{n_v} r_j}{n_v}.$$

Then, the mean interval times between packets with consecutive identical timestamps are $t_r = \frac{1}{R_r}$ (millisecond) and $t_v = \frac{1}{R_v}$ (millisecond), and the delay time is $\Delta t = t_v - t_r$. By delaying all the packet replies for Δt, the peak position in the graph of the modified run length of identical timestamp packets for a real machine coincides with the peak position for each virtual machine. However, the rate of the modified mean run lengths will concentrate around the peak position and the rate in the neighborhood of the peak position will be too low because we set the delay to be constant. To avoid such a concentration, the delay times around 15 ~ 25% of randomly selected identical timestamps in reply packets are modified to $0 \sim 0.5 \cdot \Delta t$ and $2 \cdot \Delta t \sim 4 \cdot \Delta t$. We predicted that the graph of the modified mean run length of identical timestamp packets from the real machine would almost coincide with those of the VMs. The proposed modification is shown in Figure 4.

The experiments showed that by implementing this technique, we could eliminate IP timestamps differences for a real machine and VM. That means the VM and real machines could no longer be distinguished by remotely detecting their IP timestamps.

5. Experimental Design and Measurement Methodology

The experiment was conducted on a high-performance Dell Power Edge server equipped with an Intel Xeon CPU E5-2440. XenServer, Oracle VirtualBox 4.3.12, and VMWare vSphere Hypervisor (ESXi) 5.1.0, all major commercial hypervisor products, were implemented to host the VM test machines. Open source Linux Ubuntu 12.04 with 1GB of virtual memory and IDE HDD with 16GB of virtual storage was used as the guest OS in the hypervisors and as the VM measurement targets. A machine running with Ubuntu Linux as the OS and Intel Core i3 540 as the CPU with 2.8GB of RAM was used as the real machine measurement target. A measurer machine was setup to send request packets with the IP timestamp option to the measurement target machines. The measurement target machines and the measurer machine were connected to each other via Ethernet. The experimental environment is shown in Figure 5. Each VM consisted of the minimum software base of GNU Linux running on 12.04. The utilization of each VM test environment was 80% and more.

The experiments were conducted by sending request packets with the IP timestamp option enabled in the header after the previous timestamp reply was received by the measurer machine. Although IP timestamps can be used with any type of IP packet, we only explored the attachment of the IP option and sent it with ICMP packets due to their convenient availability in the Linux ping command. We sent non-suspicious packets to the target host machine in order to make sure they would not be dropped or denied by the network or devices.

As many as 1,000,000 packets were continuously sent from the test client machine to each target machine. The packets were sent from the test client machine by executing a customized script developed for this experiment. The next request packet of the client test machine was only sent to the target host once the measurer machine had received the reply for the previous packet request. The timestamp information in the reply packets from the target machine were recorded and compiled. The IP timestamps were analyzed in decimal units to the nearest millisecond. Milliseconds was chosen as the unit for analysis as it is the standard unit for the timestamp in the IP packet [31]. Each target operating system added its timestamp, and the timestamps in the packets were not affected by the network until they reached the client machine. Thus, accurate timestamps were obtained from the target guests. Finally, the IP timestamp information from the measurement target machines was analyzed and distribution graphs were plotted in order to observe the differences between the timestamps of the target guests. After that, we implemented our timestamp modification technique on the basis of the distribution graphs and tested it to confirm its usability. A study by Kohno has proven that the clock skew is independent of the access topology, regardless of whether the hosts use random or constant IP addresses [26]. Therefore, for these experiments, we used a controlled environment with an Ethernet connection in our laboratory to eliminate the network latency issue. Note that the characteristics of the data might vary from machine to machine, from one hypervisor technology to another, and with changes in the implementation environment. Hence, we plan to conduct more tests in different experiment environments to get more data with which we can improve the implementation of the modification.

6. DATA ANALYSIS

6.1. Characteristic of IP Timestamp Patterns

The research presented in this paper are an extension to our work in [28, 29] for testing and analyzing the network timestamps discrepancies of major VM products and thus strengthens our contention that timestamps could be used for remotely detecting VM environments. In this research, we sent non-suspicious packets to the target host machines in order to make sure the packets would not be dropped or denied by the network or devices. We then analyzed the reply packets to determine whether there were any behavior differences between VMs and real machines. The results showing that there are indeed such differences supported our hypothesis that VM environments could be remotely detected by analyzing the timestamp reply patterns.

We analyzed the collected data to understand the differences between the time-stamping behaviors of the target hosts. The timestamps were extracted from the reply packets from each target host in the test client machines. Table I shows a portion of the IP timestamp data that was collected. The count information means the sequence of packets sent to the target machines. The shown sequence runs from the first packet sent, n, until the 20th packet, n + 19. In particular, for the real machine, the timestamps in the first three reply packets were the same, and those in the next five packets timestamp were also the same.

Figure 6 shows how many times the same timestamps were stamped in the sequence of reply packets from the target hosts. The percentage of identical timestamps from the real machine is obviously higher than those from the VM. This behavior occurred throughout the collected IP timestamp data. Table II shows the results of the analysis of all 1,000,000 reply packets from the real machine and VMs. We calculated the mean number of repetitions of the same IP timestamps in the reply packets from the real machine and the VMs. As shown in Table III, the mean values reveal significant differences in the IP timestamp patterns.

The real machine had a mean of almost 4.50 repetitions of the same timestamp, while the VM target hosts had smaller mean repetition values. Accordingly, we could determine that VMs and real machine had significant differences in their IP timestamp behaviors. These results support our hypothesis that since VM packets are handled via hypervisor, timestamp delays will occur in the reply packets from the VM environment. Furthermore, as expected, different hypervisor technologies had different timestamp behavior patterns. Thus, VMs could be remotely detected by using the IP timestamp pattern behavior, and each VM technology had its own mean values and pattern of timestamps. Our finding in this research are limited to the testing environment that we established in our controlled environment. More characteristic of pattern will be obtained in the future by performing more testing on various machines and implementation technologies that we plan to continue as our future work. The findings will be recorded and shared for references. Comprehensive framework will be proposed when enough data could be gathered for more comprehensive analysis.

6.2. Modification of Delay in Real Machine Environment

We performed numerical experiments to determine the effectiveness of our countermeasure of imposing delays in the real machine in order to camouflage it within the IP timestamp behavior patterns of the VMs. The experimental results, shown in Figure 7 (a), (b) and (c), indicated that the real machine could emulate the IP timestamp behaviors of the VMs. Therefore, such a modification could possibly be used as a countermeasure to prevent attackers or malware from remotely detecting VMs by exploiting the difference in IP timestamp behavior.

7. CONSIDERATIONS

In this research, we devised a way to determine whether the running environment is a VM or real one by exploiting the differences between the patterns of network timestamps and used it to detect the presence of VMs in a cloud computing environment. In our method, we sent packets that included IP timestamp option requests from the client test machine to target VM hosts. We collected the reply packets from the target hosts and analyzed the IP timestamps as to their behavior. We predicted that the pattern of reply timestamp information from a VM would be different from that of a real machine. The reason is that VMs sometimes interrupt timestamp operations to complete other operations. The time differences in the timestamps of the reply packets would thus be larger than in a real machine because a VM sometimes stops operation so that other VMs in the queue can operate and the VM clocks are managed with a timer device emulation called a VM switch.

In the experiment, the collected IP timestamp data were in milliseconds as used in the IP timestamp specification. As per our hypothesis, the real machine added timestamps more frequently than the VMs did. We camouflaged the real machine by making its IP timestamp reply pattern indistinguishable from those of the VMs. The research presented here is an extension of our research in [28] [29] to testing and analyzing major VM products and thus strengthens our conclusions in those papers. We also proposed a countermeasure wherein the timestamps in the real machine are modified to match those of the VMs. This countermeasure ensures that the VMs and real machines in the same cloud can no longer be distinguished by exploiting the IP timestamp remote detection.

8. Conclusions and Future Work

Building a transparent VM is still a difficult task; the question of how VMs can be detected is a critical one that requires extensive s research in order to prevent any security loop holes that could exploit the vulnerabilities of VMs, including security holes caused by any possible detection method. VM detection is a prelude to an attack if no countermeasure can be found. Ideally by analyzing all possible detection methods, countermeasures could be devised for making VMs more secure. Our study explored the possibility of such a detection method from the perspective of detecting the VM's existence by performing timestamp analysis in type 1 and type 2 virtualization using the IP timestamp option. This research shows that the behavior of IP timestamps in the reply packets from the most popular technique of choice, i.e., VMWare ESX, VirtualBox, and Xen, could be differentiated remotely and could be potentially used as a VM detection method. Our findings that timestamps in reply packs from VMs are different from those of real machines then was made the basis for a delay mechanism to make timestamps in the real machine look similar those of the VMs. This mechanism should make it impossible to detect whether or not the machine is a VM. In this research, we proved that VMs could be detected remotely even when they were running on a high-performance server with a Intel processor in a high-performance cloud computing environment. We also proposed a countermeasure in which a delay is imposed in the real machine in order to camouflage the IP timestamp behavior patterns. We suggest that mobile devices such as smartphones should implement delays in their timestamp stamping operations to camouflage the existence of a real environment. Our approach will decrease the chances of detection that could lead to malicious manipulation of VMs in cloud computing environment and eliminates a possible attack vector.

In the future, we will perform more studies on malware behavior and its relationship with VMs with type 1 and type 2 virtualizations. We will also develop a cloud computing security policy framework for cloud computing. In addition, we would like to test the detection method in a different environment using different machines and different implementation styles and on a grid and cloud test bed. The results could then be used to make a more accurate implementation. The findings presented here should also lead to further studies on how to provide comprehensive protection against remote detection of VM environments.

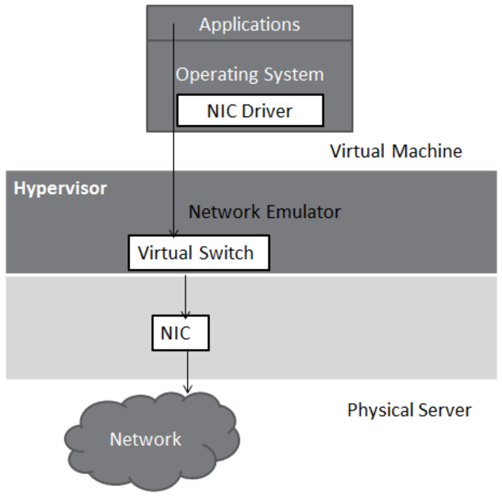

Figure 1: Virtual network path

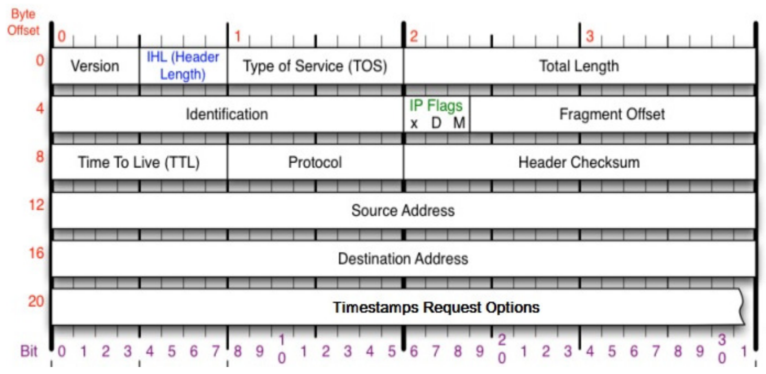

Figure 2: Data structure of IP packets

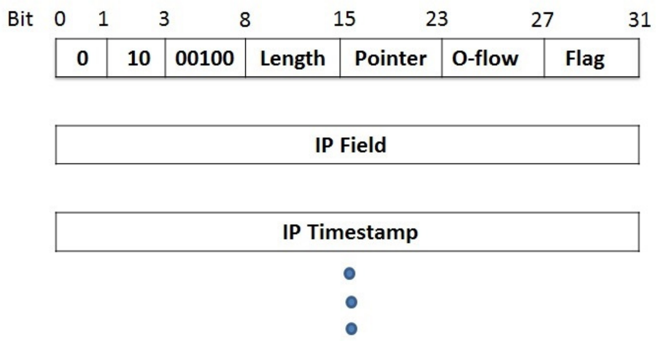

Figure 3: Structure of IP timestamp option

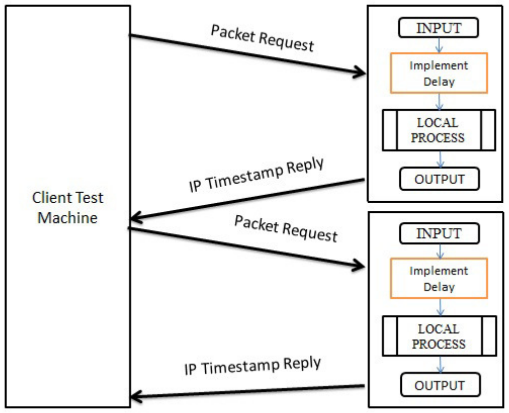

Figure 4: Proposed modification technique

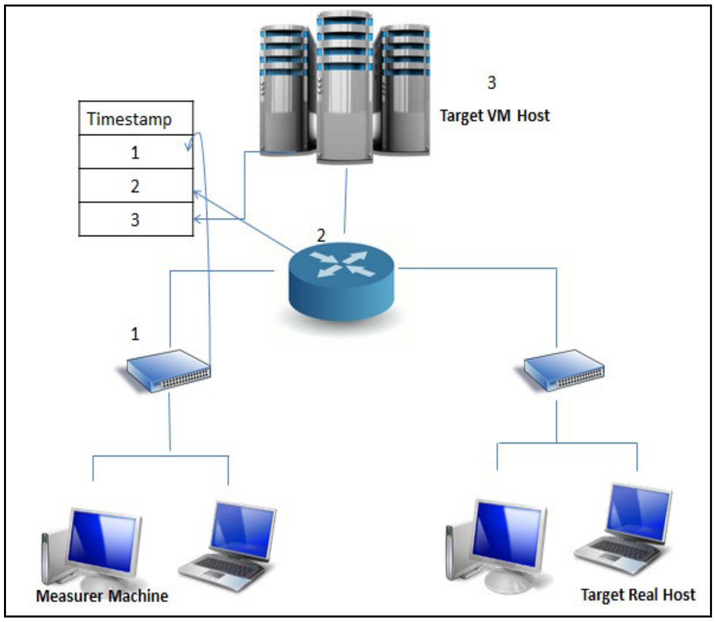

Figure 5: Experimental environment

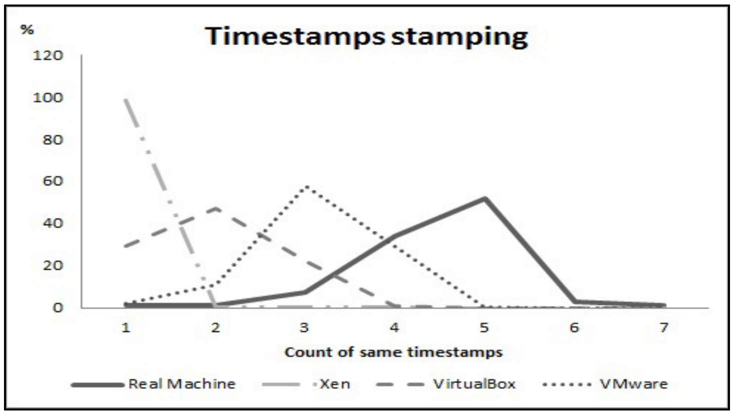

Figure 6: Analysis of IP timestamp behavior pattern of target machines

Table 1: Portion of Collected IP Timestamp Information

Count	IP Timestamp (millisecond)			
	Real Machine	Xen	VirtualBox	VMware
N	xxx 81920	xxx 38539	xxx70653	xxx77867
n+1	xxx 81920	xxx 38541	xxx70656	xxx77868
n+2	xxx 81920	xxx 38543	xxx70657	xxx77869
n+3	xxx 81921	xxx 38546	xxx70658	xxx77869
n+4	xxx 81921	xxx 38548	xxx70659	xxx77869
n+5	xxx 81921	xxx 38552	xxx70659	xxx77870
n+6	xxx 81921	xxx 38553	xxx70660	xxx77870
n+7	xxx81921	xxx38645	xxx70660	xxx77870
n+8	xxx81922	xxx38647	xxx70660	xxx77871
n+9	xxx81922	xxx38650	xxx70661	xxx77871
n+10	xxx81922	xxx38652	xxx70662	xxx77871
n+11	xxx81922	xxx38654	xxx70662	xxx77872
n+12	xxx81923	xxx38656	xxx70663	xxx77872
n+13	xxx81923	xxx38659	xxx70665	xxx77872
n+14	xxx81923	xxx38660	xxx70666	xxx77873
n+15	xxx81923	xxx38663	xxx70666	xxx77873
n+16	xxx81923	xxx38665	xxx70668	xxx77873
n+17	xxx81924	xxx38667	xxx70668	xxx77873
n+18	xxx81924	xxx38669	xxx70669	xxx77874
n+19	xxx81924	xxx38672	xxx70669	xxx77874

Table 2: Percentages of identical IP Timestamps for Real Machine and VM

How many times the same IP timestamp was stamped	Real Machine (%)	Xen (%)	VirtualBox (%)	VMware (%)
1	0.99	98.71	29.52	1.95
2	0.99	0.48	47.37	10.87
3	7.64	0.42	22.21	57.64
4	33.90	0.34	0.80	29.35
5	52.20	0.05	0.04	0.19
6	2.79	0	0.06	0
7	1.49	0	0	0

==

Table 3: Mean Number of Packets with the Same Timestamps

Host	Real Machine	Xen	VirtualBox	VMware
Mean number of repetitions of the same timestamp in the reply packets	4.50	1.03	1.95	3.15

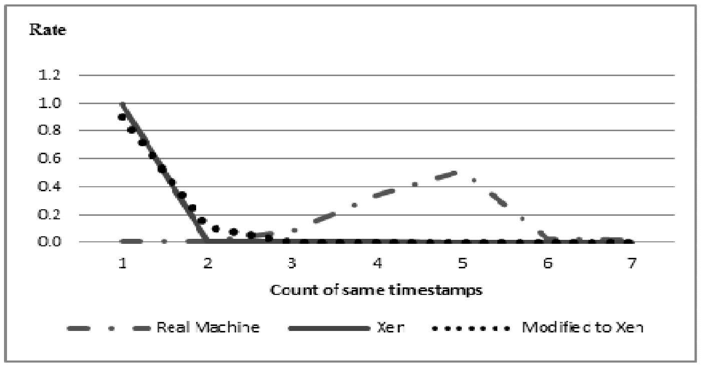

Figure 7 (a). Technique to match IP timestamp behavior of Xen

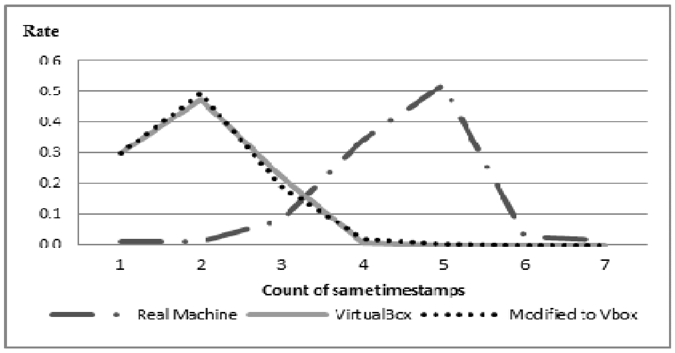

Figure 7 (b). Technique to match timestamp behavior of VirtualBox IP

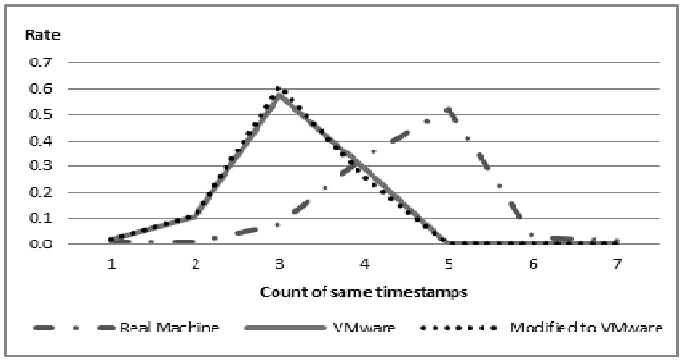

Figure 7 (c). Technique to match timestamp behavior of VirtualBox IP

REFERENCES

[1] Intelligence, G., (2014) “The Mobile Economy”.
[2] Leavitt, N., (2000) “Malicious code moves to mobile devices”, Computer, Vol. 33, Issue 12, pp16-19
[3] Rahimi, M.R., et al., (2014) “Mobile Cloud Computing: A Survey, State of Art and Future Directions”, Mobile Networks and Applications, Vol. 19(2), p133-143.
[4] Zonouz, S., et al., (2013) “Secloud: A cloud-based comprehensive and lightweight security solution for smartphones” Computer Security, Vol. 37, pp215-227.
[5] Garfinkel, T., et al., (2007) “Compatibility is not transparency: VMM detection myths and realities”, Proceedings of the 11th USENIX workshop on Hot topics in operating systems on USENIX Association, pp1-6.
[6] Ferrie, P., (2007) “Attacks on more virtual machine emulators”, Symantec Technology Exchange.
[7] Raffetseder, T., C. Kruegel, and E. Kirda, (2007) “Detecting system emulators”, Information Security on Springer, pp1-18
[8] Thompson, C., M. Huntley, and C. Link, “Virtualization detection: New strategies and their effectiveness”, http: Ilwww-users. cs. umn. edu/ cthomp/papers/vmm-detect-20.1.
[9] Younge, A.J., et al., (2011) “Analysis of Virtualization Technologies for High Performance Computing Environments”, Cloud Computing (CLOUD) on IEEE International Conference
[10] Rosenblum, M., (1999) “VMWare's Virtual Platform: A virtual machine monitor for commodity PCs”, in Hot Chips 1999.
[11] Watson, J., (2008) “VirtualBox: bits and bytes masquerading as machines”, In Linux Journal, Vol. 166, pp1.
[12] Barham, P., et al., (2003) “Xen and the art of virtualization”, Proceedings of the nineteenth ACM symposium on Operating systems principles on ACM: Bolton Landing, NY, USA, pp164-177.

[13] VMware, Inc , (2011) “Timekeeping In Virtual Machines”, https://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf.
[14] Abraham, S. and I. Chengalur-Smith, (2010) “An overview of social engineering malware: Trends, tactics, and implications”, Technology in Society, Vol. 32(3), pp183-196.
[15] Hong, J., (2012) “The state of phishing attacks”, Communications of the ACM, Vol. 55(1), pp74-81.
[16] Oberheide, J., et al., (2008) “Virtualized in-cloud security services for mobile devices”, in Proceedings of the First Workshop on Virtualization in Mobile Computing, ACM.
[17] Uda, R., (2011) “Protocol and Method for Preventing Attacks from the Web”, World Academy of Science, Engineering and Technology, Vol. 76, pp456-460.
[18] Burguera, I., U. Zurutuza, and S. Nadjm-Tehrani, (2011) “Crowdroid: behavior-based malware detection system for Android”, in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices., ACM: Chicago, Illinois, USA, pp15-26.
[19] T.Garfinkel and M. Rosenblum, (2003) “A virtual machine introspection-based architecture for intrusion detection”, in Proceedings of the Network and Distributed Systems Security Symposium, pp191–206.
[20] Portnoy, M., (2012) Virtualization essentials, John Wiley & Sons Publishers.
[21] Rutkowska, J., (2006) “Subverting VistaTM kernel for fun and profit”, Black Hat Briefings.
[22] Bahram, S., et al., (2010) “Dksm: Subverting virtual machine introspection for fun and profit”, in Reliable Distributed Systems, 29th IEEE Symposium on IEEE.
[23] King, S.T. and P.M. Chen., (2006) “SubVirt: Implementing malware with virtual machines” in Security and Privacy, Symposium on IEEE.
[24] Dunlap, G.W., et al., (2002) “ReVirt: Enabling intrusion analysis through virtual-machine logging and replay”, ACM SIGOPS Operating Systems Review, Vol. 36(SI), pp211-224.
[25] Fonseca, R., et al., (2005) “IP options are not an option”, Univ. of California, Berkeley.
[26] Kohno, T., (2005) “Remote physical device fingerprinting”, IEEE Transactions on Dependable and Secure Computing, Vol. 2(2), pp93.
[27] Shimamura, M., Kono, K., (2009) “Remote Virtual Machine Monitor Detection Using Network Timestamp”, IPSJ, Vol. 50(8) (Japanese), pp1870-1882.
[28] M. Noorafiza, et al. (2013) “Virtual Machine Remote Detection Method Using Network Timestamp in Cloud Computing”, Proceedings of the Information Science and Technology (ICIST) on IEEE.
[29] M. Noorafiza, H.Maeda., R. Uda, T. Kinoshita, M. Shiratori, (2015) “Vulnerability Analysis using Network Timestamps in Full Virtualization Virtual Machine”, in 1st Proceedings of International Conference on Information Systems Security and Privacy (ICISSP 2015), SCITEPRESS Digital Library
[30] Mills, D., (1992) “Network Time Protocol (Version 3) specification, implementation and analysis”.
[31] Su, Z., (1981) “Specification of the Internet Protocol (IP) Timestamp Option”.

AUTHOR

Noor Afiza received her Bachelor of Engineering in Information and Computer Sciences from Ibaraki University in 2002 and Master of Science in Computer Science from Tokyo University of Technology in 2013. She is PhD candidate in Tokyo University of Technology.

Hiroshi Maeda received his Bachelor of Science and Master of Science in Computer Science from Tokyo University of Technology in 2012 and 2014 respectively. He is specializing in Information Security Research.

Toshiyuki Kinoshita received his Bachelor from Tokyo Institute of Technology in 1975, Master of Science from Tokyo University in 1977 and worked for Systems Development Laboratory, Hitachi Ltd. for 28 years. He received his Ph.D. of Science from Tokyo Institute of Technology in 2000 and became Professor in Tokyo University of Technology in 2005. Kinoshita is majoring in Computer Security, System Programing and Queuing Theory. He is also a member of IPSJ and ACM.

Ryuya Uda received his Bachelor of Engineering, Master of Engineering and Ph.D. of Engineering from Keio University in 1998, 2000 and 2002 respectively. After that, he was working as Research Assistant in Tokyo University of Technology and promoted to Assistant Professor in 2003. Uda is Specialist in Network Security and won IFIP SEC Best Student Paper Award in 2002 and also a member of IPSJ.