Rollback Mechanism of Nested Virtual Machines for Protocol Fuzz Testing

Kuniyasu Suzaki*, Toshiki Yagi, Akira Tanaka*, Yutaka Oiwa*, Etsuya Shibayama*,‡

{k.suzaki, yagi-toshiki, tanaka-akira, y.oiwa}@aist.go.jp, etsuya@ecc.u-tokyo.ac.jp

* National Institute of Advanced Industrial Science and Technology(AIST)
Research Institute for Secure Systems (RISEC)
Tsukuba Central 2, 1-1-1 Umezono, Tsukuba,
Ibaraki, 305-8568, Japan

‡ The University of Tokyo
Information Technology Center
2-11-16 Yayoi, Bunkyo,
Tokyo 113-8658, Japan

ABSTRACT

Secure communications (HTTPS, SSH, etc) are important in the current Internet services. Implementations of secure protocols should be tested as exhaustively as possible. Repeated protocol fuzz testing from every reachable state is necessary and snapshot/rollback mechanism is required. Ordinary snapshot tools, however, only bring back a state of process or virtual machine (VM), and do not take care of packets on a wire. It means that they have no feature of distributed snapshot defined by Chandy-Lamport. Furthermore, secure protocols inherently depend upon a computing environment (e.g., random number) and make it difficult to repeat same testing. In order to solve these problems easily and generally, we propose a new protocol for controlling snapshot/rollback of VM, and an implementation which uses nested VMs and proxies. The internal VM of nested VM emulates whole hardware for exact repeat of protocol handling, and the external VM and proxies work for managing the state of internal VM and packets on a wire. In the current implementation internal VM is the instruction emulator QEMU and external VM is KVM which uses virtualization instructions. On a feasibility study, 4 TLS 1.2 servers (OpenSSL, GnuTLS, CyaSSL, and PolarSSL) were verified, and we found 2 bugs in CyaSSL and 1 bug in PolarSSL.

Categories and Subject Descriptors

C.4 [Computer Systems Organization]: Performance of Systems—Measurement Techniques; C.2.2 [Computer Systems Organization]: Network Protocols — Protocol verification

General Terms

Design, Verification, Measurement, Performance, Reliability, Security

SAC'14, March 24-28, 2014, Gyeongju, Korea.

Keywords

Protocol Fuzz testing, Nested Virtual Machines, Virtual Machine Monitor, Snapshot, Rollback, Transport Layer Security (TLS), KVM, QEMU, OpenSSL, GnuTLS, CyaSSL, and PolarSSL

1. INTRODUCTION

In recent years, secure communications (HTTPS, SSH, etc) become important because most of the Internet services depend on them. Unfortunately, vulnerabilities of secure communications are discovered repeatedly. For example, TLS certificates created by random numbers with insufficient entropy came to significant issues in 2012 [5,11,13]. Many same TLS certificates were distributed and frightened the root of trust. In order to settle the problem, Electronic Frontier Foundation (EFF) offers a site which investigates correctness of TLS certificates [6]. It shows that implementations of secure protocols should be tested as exhaustively as possible.

Fuzz testing of protocol is popular software test technique that provides invalid, unexpected, or random packets to an implementation of protocol [9]. Current brute-force fuzz testing, however, is not enough for an implementation of secure protocol because secure protocol has states to change mode (e.g., negotiating from plain text to cipher), which is not easily reached with randomly created packets. Implementation of changing mode is likely to make a mistake and should be tested repeatedly with sophisticated fuzz packets. In order to make precise protocol fuzz testing, we are developing a fuzz packet generator which creates test packets based on a specification of protocol (i.e., RFC of IETF) and investigates implementations of critical protocol handling repeatedly.

Repeated protocol fuzz testing requires same repetition of protocol handling. However, standard snapshot does not satisfy the requirements. First, ordinary snapshot tools treat only one computing instance (i.e., a process or a virtual machine (VM)) and do not treat multi computing instances in general. The feature may cause inconsistency between a server and client because it does not take care of packets on a wire. This problem is known as distributed snapshot defined by Chandy-Lamport[2]. Second, the

protocol fuzz testing requires taking snapshot just before sending a certain packet. However, most snapshot tools are independent of packet, and they must be customized to be triggered by a certain packet. Third, some secure protocols inherently depend upon a computing environment (e.g., random number generated by hardware; Intel Sandy Bridge's RDRAND instruction, random generator in TPM (Trusted Platform Module) chip, etc). The random numbers should be exactly repeatable because they are used to find vulnerabilities on a certain computing status. It means that rollback schemes for secure protocol test have to treat whole computing environment.

In this paper, we propose a new protocol for controlling snapshot/rollback of VM. The protocol encapsulates packets of target protocol in order to place them under control. As an implementation of the protocol handler, we propose to use nested VMs with proxies. The internal VM of nested VM emulates whole hardware for exact repeat of protocol handling, and the external VM and proxies work for managing the state of internal VM and packets on a wire. The implementation is not only easy but also general to apply other VMs because it utilize an existing snapshot/rollback function of standard virtual machine monitor (VMM) and does not requires any customization of VMM. Nested VMs may seem to be over-specification, but some optimizations (reducing memory usage and omitting GUI) achieve appropriate performance. The feasibility of nested VMs was confirmed for 4 TLS 1.2 servers (OpenSSL, GnuTLS, CyaSSL, and PolarSSL). They helped to find 2 bugs in CyaSSL and 1 bug in PolarSSL.

This paper is organized as follows. Section 2 mentions related works and Section 3 describes the detail of difficulty of protocol fuzz testing. Section 4 presents new protocol which encapsulates existing secure protocol and control VM snapshot and rollback. Section 5 describes the design of rollback scheme which uses nested VMs and proxies. Section 6 reports the current implementation and its performance on the fuzz packet generator for TLS 1.2. Section 7 discusses future works and Section 8 summarizes this paper.

2. RELATED WORKS

The requests for rollback scheme on protocol fuzz testing seem to be same for VM live migration or VM debugging. However, VM live migration and VM debugging are based on a different assumption and do not cause problems.

VM live migration [4,14,16] can guarantee that time does not go back and does not need to care of packets controlled by TCP because they are re-send by TCP mechanism. The point is that VM live migration is caused by one-side VM (server or client), but protocol fuzz testing assumes that rollback causes both server and client.

Normal VM debugging [3,8,12,20] treats only one guest OS and keeps a log of precise hardware behavior. The problem of VM debugging is not the keeping a log. The problem is the understanding of meaning of hardware usage of guest OS, which is known as VM introspection[15,18]. On the other hand, protocol fuzz testing need not to know the detail of OS behavior, but has to maintain consistency between a server and client because normal snapshots are taken on the server and client separately. The same situation happens when a guest OS runs on multi-processors. The problem is treated by synchronization in SMP-ReVirt [7] or serialization in DoublePlay [19] on VMM. They require special customization of VMM. On the other hand, our snapshot/rollback scheme is implemented on nested VMs and proxies, and does not require customization of VMM.

Nested VMs may seem to be over-specification because it uses much computing resource and causes performance degradation. However, continuous studies of nested VMs [1,10] reduce overhead, and some application can use nested VMs to solve individual problem. For example, CloudVisor[22] uses nested VMs for security protection in multi-tenant cloud. Xen-Blanket[21] uses nested VMs for VM live migration on Amazon EC2.

3. DIFFICULTY OF PROTOCL FUZZ TESTING

Protocol fuzz testing requires exact repeat of protocol handling to test an implementation of secure protocol. The implementation of fuzz testing, however, has some problems.

3.1 Consistency between a Server and Client

In order to resume secure communication, the consistency between a server and client has to be kept. Ordinary snapshot tools, however, take a snapshot image of a process or a VM instance on one side (client or server), and do not take care of the packets on a wire, which is known as distributed snapshot problem [2]. It does not mean the simultaneous snapshots on a server and client because the time to take snapshot is usually longer than the time to send a packet, and it is not easy to guarantee physical coincident snapshots on distributed systems. The point is to keep logical consistency of already-created packets.

In order to deal with packets on a wire, we use control protocol proposed in section 4. A control packet make a round-trip on a wire between the server and client, and the snapshot mechanism confirms no packets on a wire. The detail is described in section 4.

3.2 Packet Level Granularity Control

Protocol fuzz testing for secure protocol requires to repeat tests for each packet. It means that snapshot has to be managed by packet level granularity. A snapshot command has to be issued before a certain packet.

Unfortunately, current snapshot tools cannot distinguish packets because snapshot command is issued independent of packet. To solve this problem, snapshot mechanism needs to recognize each packet.

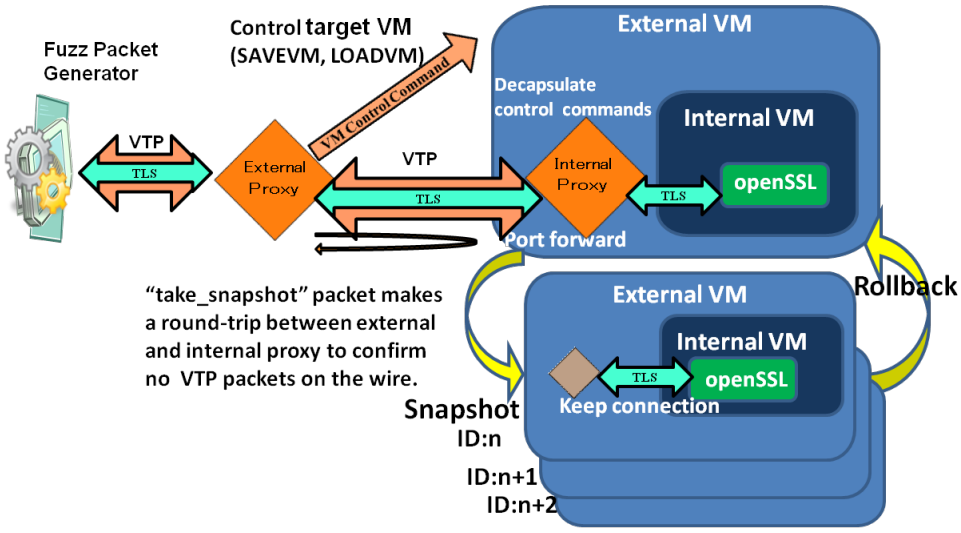

Figure 1. Implementation of packet level snapshot/rollback using nested VMs and proxies

Furthermore, a connection between the server and client is cut when each snapshot is taken on a certain virtual machine. The rollback scheme needs to manage the reestablishment of the connection when it brings back the status.

As a solution, we propose protocols which control snapshot and rollback. A test-target protocol is encapsulated by this protocol at a fuzz packet generator and de-capsulated by a snapshot tool on a test-target. Using this protocol, snapshot is taken before a certain packet in the stream of secure communication. When a snapshot is resumed, the protocol handler has to reestablish the network connection instead of virtual machine.

3.3 Rollback with Computing Environment

Many implementations of secure protocol depend on a computing environment. When the computing environment is changed, the behavior of secure communication is also changed. For example, TLS certificates created by random numbers with insufficient entropy came to big issues in 2012 [5,11,13]. Ordinary random number generator is not included in an implementation of secure protocol because random numbers are offered by its computing environment. Some random number generators depend on hardware (e.g., Intel Sandy Bridge's RDRAND instruction) and are not repeatable. This evidence indicates the protocol testing should include its computing environment.

For protocol fuzz testing, rollback of computing environment should be repeatable. It means all behavior should be repeated exactly. If a rollback image behaves differently, it makes difficult to verify the implementation. Therefore, random numbers must be repeatable, when rollback is occurred.

In order to include repeatable random numbers, we need to use a virtual machine which emulates full hardware. However, the repeatable random numbers on a virtual machine also caused another problem on real usages[17]. The repeatable random numbers give a hint to an attacker. Some virtual machine has a mechanism to prevent repeatable random numbers. Therefore, we have to choose a virtual machine which repeats same random numbers for protocol test.

4. PROTOCOL WHICH ENCAPSULATES EXISTING PROTOCOL AND CONTROLS VM

We propose a new protocol called VTP (Virtual Test Protocol) that encapsulates existing secure protocol and controls VM snapshot and rollback. VTP maintains secure communication streams on a wire. VTP consists of mainly three elementary protocols; "Capsule", "Take_Snapshot", and "Rollback". The characters are summarized in Table 1. VTP is designed for our fuzz packet generator but can be used for other purposes.

Figure 1 shows a scenario of VTP with nested VMs and proxies. Our fuzz packet generator verifies TLS protocol on the environment.

Table 1. VTP (Virtual Test Protocol) which encapsulates existing protocol and controls VM

Protocol	Action
Capsule	Packets of secure communication are encapsulated with capsule protocol. The flow of encapsulated packets is managed when take_snapshot or rollback packet is issued.
Take_ Snapshot	It takes VM snapshot image. Packets on the wire must be managed by a certain method. Return snapshot ID.
Rollback (with ID)	It resumes snapshot. Network connection is reestablished if VM disconnects the communication.

All TLS packets issued by fuzz packet generator are encapsulated by "Capsule" protocol. The packets are de-capsulated at the internal proxy. Each capsulated packet is treated as data packet. The packet has a sequence number, and the flow is controlled. The original TLS protocol does not concern to the physical condition of network.

A take_snapshot packet is used to take a snapshot of external VMs and returns the snapshot ID. It also confirms all encapsulated packets are flushed from the wire. The flush is confirmed by round-trip of take_snapshot packet between external proxy and internal proxy, which is same to consistent global state defined by Chandy-Lamport[2].

A rollback packet with a snapshot ID resumes the corresponding snapshot image of external VM. The rollback packet manages dis- and re-connections without being recognized by the secure communication. The detail of VM management is described in the next section.

Taking a snapshot and rollback of external VM are achieved by the function offered by standard VMM. This implementation does not require any customization and can be applied on any virtual machine monitors, which allows nested VMs.

5. NESTED VMS AND PROXIES

Protocol fuzz testing requires exact repeat of protocol handling, and protocol level control of snapshot. In order to achieve these requirements easily, we use nested VMs and proxies. The internal VM of nested VM emulates whole hardware for exact report of protocol handling, and the external VM and proxies work for managing the state of internal VM and packets on a wire. External VM does not need to emulate whole hardware. It is rather required fast control of internal VM.

As a prototype implementation, Nested VMs is consisted of KVM and QEMU being external and internal VMMs, respectively. Internal QEMU is a software emulator and repeat same hardware behavior when it is rolled back by external KVM. External KVM uses virtualization instruction (e.g., Intel VT-d and AMD SVM) and achieves good performance. As described in Figure 1, an internal proxy runs on the external KVM to manage the proposed protocol (VTP), and an external proxy runs on the host OS to manage snapshot of external KVM which includes internal QEMU.

The following sections describe the detail of protocol handling for snapshot and rollback and optimization for nested VMs.

5.1 Protocol Handling

The standard snapshot function on a VM does not maintain the network connection. This problem is solved with nested virtual machines (external VM and internal VM) and proxies (Figure 1). An external VM includes an internal VM and an internal proxy. The internal VM must be repeat full computing environment (i.e., random number). The internal proxy connects to an internal VM. When a take_snapshot packet is sent, external VMM takes a snapshot, which keeps the state of the connection between the internal VM and the internal proxy. Even if the external VM is resumed to a snapshot image with rollback packet, the connection is operative. However, the connection to the outside (external proxy) is not included in the snapshot. The external proxy recognizes the rollback packet and reestablishes the connection.

Figure 2 shows the implementation detail of proxies. The two ports are used to control the external VM. One port (12344) is

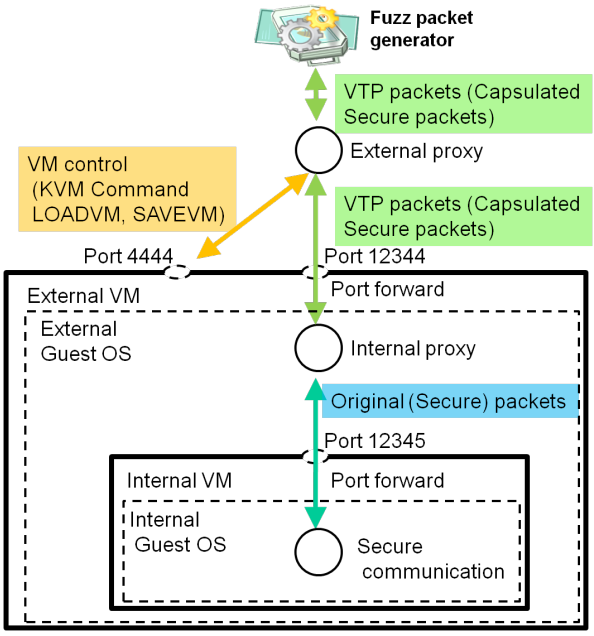

Figure 2. Implementation detail of external and internal proxies on nest VMs.

used for encapsulated packets, and the other port (4444) is used for VM control commands (SAVEVM is a command to take a snapshot and LOADVM is a command to rollback an image). For internal VM, only one port (12345) is used for encapsulated packets because the internal VMM never take a snapshot.

All packets for secure communication are encapsulated and de-capsulated at internal proxy. When a take_snapshot command is issued at the external proxy, the take_snapshot packet is sent to the internal proxy. When the packet is arrived at the internal proxy, internal proxy disallows any write requests from internal VM using "select" of Perl. It means that a write request from internal VM is suspended, but the connection is still alive. Internal proxy sends back take_snapshot packet to confirm all encapsulated packets are flushed between internal proxy and external proxy. After the confirmation, external proxy sends SAVEVM command with an ID to the port 4444 which connects to the external VMM. The external VMM takes its snapshot with the ID. The ID is managed by the external proxy. After that, the connection between external proxy and internal proxy is disconnected.

When the fuzz packet generator issues a rollback packet with snapshot ID, the external proxy sends LOADVM command to the port 4444. The external VMM resumes corresponding snapshot. At that time, the connection between external and internal proxy is still disconnected. After resuming snapshot image, external proxy sends rollback packet to internal proxy. When the internal proxy receives a rollback packet, the internal proxy reestablishes the connection between external and internal proxy. After that it allows write requests from internal VM again using "switch" of Perl.

5.2 Optimization

Most virtual machines have a default GUI which emulates a video card. In general nested VMs emulate the GUI of internal VM on external VM again, which consumes much time. Fortunately QEMU and KVM have "curses mode" which emulates text user interface and displays VGA output. In our implementation, we nests curses mode on internal and external VMs.

When GUI is removed, guest OS on internal and external VM can delete X window and window manager, and reduce consumption of memory. It means the reduction of snapshot overhead because most snapshot commands save whole memory image of VM to a file. The effects are mentioned in performance evaluation section.

6. IMPLEMENTATION AND FEASIBILITY STUDY

The prototype of nested VMs (external KVM and internal QEMU) uses Debian squeeze 6.0.5(Linux 2.6.32) as a guest OS. The internal and external proxies are implemented in Perl, and their sizes are small (External is 430 LOC, Internal is 132 LOC).

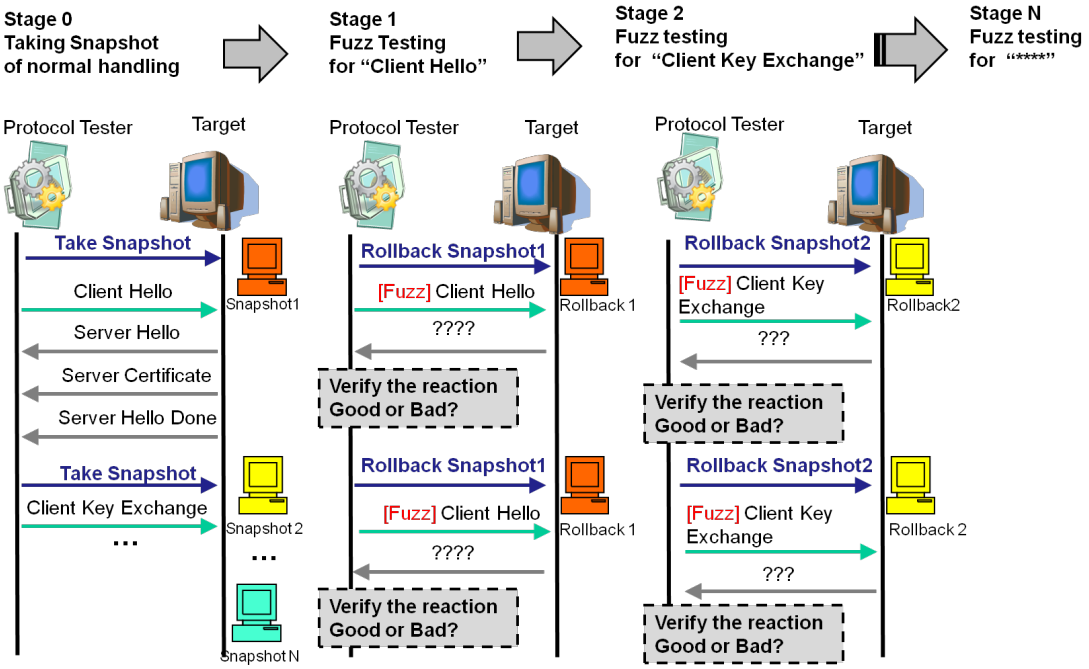

Figure 3. Process of fuzz packet generator for TLS.

6.1 Fuzz Testing on TLS Servers

We applied the fuzz packet generator to nested VMs. Figure 3 shows the process of the fuzz packet generator which tried to check handshake of TLS 1.2 protocol on 4 TLS servers (OpenSSL, GnuTLS, CyaSSL, and PolarSSL).

At first (stage 0 in Figure 3), the fuzz packet generator sends valid packets to check valid reaction. The fuzz packet generator takes a snapshot before sending a valid packet because the snapshot is used for fuzz testing. The No.1 snapshot is taken before fuzz packet generator sends "client hello". After that fuzz packet generator receives correct "server hello, Server Certificate, and Server Hello Done". Before sending "Client Key Exchange", the fuzz packet generator takes No.2 snapshot. It continues the end of the valid communication.

After finishing the valid communication, the fuzz packet generator starts fuzz testing. The fuzz packet generator resumes the No.1 snapshot image, and sends a fuzz packet of "client hello" in order to check the reaction (stage 1). After the first fuzz packet, the fuzz packet generator resumes again the No.1 snapshot image and sends another fuzz packet. It continues the end of fuzz testing for "client hello" protocol.

After the fuzz testing for "client hello" protocol, the fuzz packet generator resumes the No.2 snapshot image and sends a fuzz packet "Client Key Exchange" (Stage 2). It also continues the end of fuzz testing for "Client Key Exchange" protocol. The sequence of packet fuzz testing continues to the end of handshake of TLS protocol.

The fuzz testing found 2 bugs in CyaSSL.

- ✓ When a client sends corrupt finished message to CyaSSL, CyaSSL returns encrypted alert message before sending ChangeCipherSpec.
- ✓ When a client sends HelloRequest message to CyaSSL, CyaSSL returns "no_renegotiation" alert.

It also found following 1 bug in PolarSSL.

- ✓ Handshake fails by incorrect CertificateRequest message.

We reported the results to the mailing list, and the bugs were fixed.

Table 2. Time for fuzz testing (Internal VM is QEMU (PolarSSL on Linux), External VM is KVM).

		GUI Mem: 1024/512 (sec)	NoGUI Mem: 1024/512 (sec)	NoGUI Mem: 512/256 (sec)
Setting up nested VMs		266	107	93
Fuzz packet generator		1,307	1,164	1,080
Nested VMs and Proxies	Snapshot (9 times)	57 (6.33)	37 (4.11)	24 (2.67)
	Rollback (2,311times)	3,135 (1.36)	2,197 (0.96)	1,286 (0.56)
	Other	12	13	12
Total		5,043	3,622	2,587

6.2 Performance

The performance was evaluated on the environments with and without GUI and on large memory (1GB for external VM, and 512MB for internal VM.) and small memory (512 MB for external VM, and 256MB for internal VM). The target TLS server was PolarSSL, but other TLS servers (OpenSSL, GnuTLS, CyaSSL) showed almost same performance. The test bed machine was ThinkPad T410 (CPU Intel Core i7-M620 2.67Ghz, Memory 4GB).

Table 2 shows the time for each fuzz testing of TLS handshake. Fuzz packet generator took 9 snapshots and used them 2,311 times for rollback. The results show that NoGUI and small memory increased the performance. The most influential factors were snapshot and rollback. NoGUI reduced them to 70% and GUI and small memory reduced them to 40%. The total time could be half from GUI with large memory to NoGUI with small memory. As the result, the overhead time caused by nested VM and proxy (1,286 sec) became almost same to the time consumed by fuzz packet testing on NoGUI and small memory environment (1,080 sec).

7. DISCUSSIONS

The performance (time and image size) of take_snapshot and rollback depends on the size of memory used in a VM. It is caused by taking whole memory image for snapshot. If a VMM can use a differential snapshot mechanism which saves updated memory pages only, the overhead time will be reduced. Same technique is used on VM migration [16] but is not yet applied on VM snapshot. Especially, a fuzz packet generator takes a snapshot for each packet with very short time period, and the updated memory pages are expected to be very few.

Easy and extensible implementation scheme for testing tool is important. For example, our proposed method does not depend on a special combination of nested VMs and allows replacing internal VM with other CPU emulator. We also tried ARM QEMU as internal VM and also verified three TLS 1.2 servers (OpenSSL, GnuTLS, and CyaSSL). It shows the nested VMs are flexible and can be applied on other situations (e.g., I/O fuzz testing, memory fuzz testing, API fuzz testing, etc).

8. CONCLUSIONS

The implementations of secure protocol should be tested as exhaustively as possible. Protocol fuzz testing is desired, but it requires packet level control of snapshot, and exact repeats of secure communication which includes random number.

In order to solve these problems, we proposed a new protocol for controlling snapshot/rollback of VMs and for encapsulating packets of the target protocol. We also proposed a simple implementation scheme of snapshot/rollback mechanism with nested virtual machines and proxies.

The scheme was implemented on the combination of external KVM and internal QEMU virtual machines. The implementation was used for a real fuzz packet generator, which automatically creates sophisticated fuzz packets based on a specification of test target protocol (i.e., RFC of IETF). The handshake of TLS 1.2 protocol on 4 TLS servers (OpenSSL, GnuTLS, CyaSSL, and PolarSSL) were verified and found 2 bugs in CyaSSL and 1 bug in PolarSSL. The bugs are reported to developer's mailing lists and fixed. The verifications also showed overhead of nested virtual machines could be half by omitting GUI and small memory size for the guest OS, which was almost same time consumed by protocol fuzz packet generator.

The proposed implementation scheme utilizes existing snapshot/rollback function of virtual machine monitor and does not require any customization. The scheme can be applied on other CPU emulator and verify protocol handlers on other CPU architecture.

ACKNOWLEGEMENT

This work is supported by the National Institute of Information and Communications Technology of Japan.

REFERENCES

[1] M.Ben-Yehuda, M.D. Day, Z. Dubitzky, M.Factor, N.Har'El, A. Gordon, A. Liguori, O. Wasserman, B. Yassour, The Turtles project: Design and implementation of nested virtualization, 9th Symposium on Operating Systems Design and Implementation (OSDI), 2010.

[2] K.M.Chandy and L. Lamport, Distributed Snapshots: Determinig Global States of Distributed Systems, ACM Transactions on Computer Systems, Vol.3, No.1, 1985.

[3] E Christopher Lewis, Prashant Dhamdhere, and Eric Xiaojian Chen, Virtual Machine-Based Replay Debugging, Google Tech Talk, http://www.youtube.com/watch?v=RvMlihjqlhY

[4] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, Live Migration of Virtual Machines, Symposium on Networked Systems Design and Implementation (NSDI), 2005.

[5] P. Eckersley and J. Burns, Is the SSLiverse a Safe Place?, DefCon, 2012.

[6] EFF SSL Observatory, https://www.eff.org/observatory

[7] G.W. Dunlap, D.G. Lucchetti, P.M. Chen, and M.A. Fetterman, Execution Replay for Multiprocessor Virtual Mchines, Virtual Execution Environments (VEE), 2008.

[8] G.W. Dunlap, S.T. King, S. Cinar, M. Basrai, and P.M. Chen, ReVirt: enabling intrusion analysis through virtual-machine logging and replay, 5th Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[9] M. Greene, A. Amini, and P. Sutton, Fuzzing: Brute Force Vulnerability Discovery, Addison-Wesley, 2007

[10] Q. He, Nested Virtualization on Xen, Xen Summit Asia, 2009.

[11] N. Heninger, Z. Durumeric, E. Wustrow and J.A. Halderman, Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices, USENIX Security Conference, 2012.

[12] S.T. King, G.W. Dunlap, and P.M. Chen, Debugging operating systems with time-travelling virtual machine, USENIX Annual Technical Conference, 2005.

[13] A.K. Lenstra, J. P. Hughes, M. Augier, J.W. Bos, T. Kleinjung, and C. Wachter, Ron was wrong, Whit is right, Crypto, 2012.

[14] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu., Live Virtual Machine Migration with Adaptive Memory Compression, High performance Distributed Computing (HPDC), 2009.

[15] K. Nance, B. Hay and M. Bishop, Virtual Machine Introspection Observation or Interference?, IEEE Security & Privacy, September/October, 2008.

[16] M. Nelson, B.H. Lim, and G. Hutchins, Fast Transparent Migration for Virtual Machines, USENIX Annual Technical Conference, 2005.

[17] T. Ristenpart and S. Yilek, When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography, Network and Distributed System Security Symposium (NDSS), 2010.

[18] T. Garfinkel M. Rosenblum, A Virtual Machine Introspection Based Architecture for Intrusion Detection, Network and Distributed System Security Symposium (NDSS), 2003.

[19] K. Veeraghavan, D. Lee, B. Wester, J. Ouyang, P.M. Chen, J. Flinn, and S. Narayanasamy, DublePlay: Parallelizing Sequential Logging and Reply, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) 2011.

[20] VMware, Replay Debugging on Linux, VMware Technical note.

[21] D. Williams, H. Jamjoom, and H. Weatherspoon, The Xen-Blanket: Virtualize Once, Run Everywhere, European Conference on Computer Systems (EuroSys), 2012.

[22] F. Zhang, J. Chen, H. Chen and B. Zang, CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization, 23rd Symposium on Operating Systems Principles (SOSP), 2011.