Mem-Shot: 악성코드 난독화 분석을 위한 API-Trigger 기반의 메모리 덤프 시스템 설계 및 구현

Mem-Shot: Design and Implementation of API-Trigger Driven Memory Dump System for Obfuscated Malware Analysis

최상훈, 박기웅[1)]
Sang-Hoon Choi, Ki-Woong Park

(32588) 충청남도 공주시 공주대학로56 공주대학교 산학협력단
(05006) 서울특별시 광진구 능동로 209 세종대학교 정보보호학과
csh0052@gmail.com, woongbak@sejong.ac.kr

요 약

최근 유포되는 악성코드에는 악성코드 분석을 방해하기 위한 코드삽입, 난독화, 문자열 암호화 등 다양한 악성코드 분석회피 기술이 적용되고 있다. 본 논문에서는 이러한 분석회피 기술이 적용된 악성코드의 분석을 위해, 가상머신에 악성코드를 구동시킨 후 악성코드가 특정 API를 호출하면 정확한 시점에 가상머신의 메모리 이미지를 빠르게 추출 할 수 있는 악성코드 분석 시스템을 구현하였다. 악성코드에서 특정 API가 호출된 정확한 시점에 메모리 덤프 파일을 얻을 수 있다면, 메모리분석을 통해 악성코드가 사용한 함수의 매개변수나 암호화 된 데이터 및 난독화가 해제된 코드 정보를 얻을 수 있게 된다. 실험결과 악성코드가 API호출한 정확한 시점에 메모리 덤프를 할 수 있었고, 메모리 분석을 통해 분석회피 기술이 적용된 악성코드로부터 숨겨진 문자열과 API 매개변수를 추출 할 수 있었다.

Abstract

As malware generation techniques have been advanced, malware authors utilize various malware analysis evasion techniques such as obfuscation, garbage code insertions and string encryption. To alleviate such problems, we designed and implemented a malware analysis system which is specialized in dumping memory of a virtual machine. Malware analysis based on memory dump is a promising way to deep dive into the obfuscated malwares. Our system makes it possible to take a memory snapshot at a time of a certain API called. Furthermore, it accelerated the memory dump. Consequently, users can extract hidden information such as encrypted data and functional parameters from the malware in a user friendly manner. According to our experiments, our system can detect such hidden strings and API arguments even with analysis evasion techniques.

※ 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(2016R1A4A1011761).
1) 교신저자

논문투고일자 : 2016. 08. 01 심사일자 : 2016. 08. 01 게재확정일자 : 2016. 08. 22

키워드: 악성코드, 메모리분석, 가상화 기술
Keyword: Malware, Memory Analysis, Virtualization

1. Introduction

As malware obfuscation software has been widely popularized, the number of malware equipped with obfuscation techniques such as garbage code insertions and string encryption has increased [1]. To analyze the malwares, diverse malware analysis techniques have been used. Among them, malware analysis based on memory dump is a promising way to deep dive into the obfuscated malwares. In an effort to analyze malware behavior from the memory dump, many malware analysts have hoped to acquire a memory dump on a certain time. To achieve it, two main approaches which are instruction-driven memory dump and periodical memory dump have been used. However, the above two approaches have critical defects from the performance perspective or from the exactness perspective. For example, instruction-level memory dump causes user-obstructive latency whenever a certain instruction is executed. Furthermore, malware authors use code obfus- cation technique such as Themida [2] or put the garbage code which confuses malware analysts. On the other hand, periodical memory dump has low overhead, but it is possible to miss important data in malicious code because code obfuscation techniques may hide them after execution of the malware.

As a remedy to the problems, Tomer Teller [3] has announced API-trigger driven memory analy-sis scheme to extract unpacked code and API parameters based on Cuckoo Sandbox [4]. Although Tomer Teller' s approach is effective, it has a limitation to apply advanced malware analysis techniques on the Cuckoo Sandbox patched by Tomer Teller because the source code of the dll is not disclosed. In this paper, we incorporated a conventional Cuckoo Sandbox into our system as an underlying memory dump framework in an attempt to provide a way of publically opened for additional customization of Cuckoo Sandbox.

As a preliminary experiment, we performed memory dump in response to a certain API call. As shown in Fig. 1, we found that the time gap exists between the time of the API call and the time of the memory dump time. For example, on the reception of API-triggering event, Cuckoo Sandbox receives the message from the guest OS. Within this time gap, there is still a possibility of hiding malicious code and data again. To perform the API trigger-based memory dump at a certain time, we revised a framework for dumping memory of guest virtual machine by profiling the conventional Cuckoo Sandbox. As a next step, we successfully removed the time gap between the API call and dump. As a result, we can extract the string to be decoded only at specific times.

This study is an extension of our previous work[5], in which we focused on systems that can quickly dump memory at the time of the certain API calls. Our objective in this study however, is to devise a memory dump and integrate the overall memory analysis in auto malware analysis system.

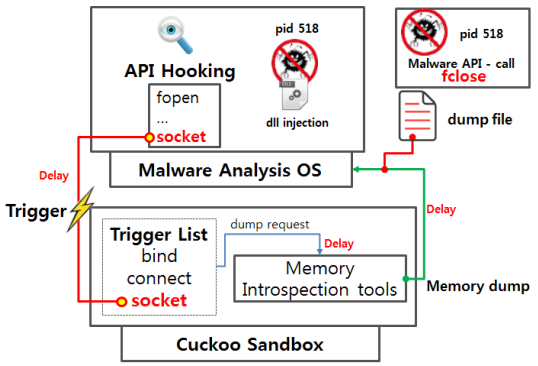

Fig. 1 API trigger-based memory dump On Cuckoo Sandbox

This paper is organized as follows. In section 2, we introduce the related work that performs API trigger-based memory dump technology and memory dump tools. In section 3 we will present our research motivation and problems to solve by analyzing and profiling Cuckoo Sandbox. Additionally, we explain the design and the implementation to solve the problems. In section 4, we describe the performance comparison the existing Cuckoo Sandbox with API trigger and memory dump times. Finally, in section 6, we present our conclusion in this work.

2. Related Work

2.1 Memory Dump Tools

- **LibVMI**: LibVMI [6] is a C library with Python bindings that makes it easy to monitor the low-level details of a running virtual machine by viewing its memory, trapping on hardware events, and accessing the vCPU registers. LibVMI makes it possible to dump the memory of the virtual machine. However LibVMI causes user-obstructive latency for acquiring a memory dump from the virtual machine (15,254, 220ms).
- **Libvirt**: Libvirt [7] is a library tool for managing the virtual machine. Libvirt provides APIs for creating, controlling, migrating virtual machines. For instance, *dump*() is to obtain a memory dump file from the virtual machine. During the execution time of *dump*(), the virtual machine is temporally stopped due to the extraction operation for acquiring memory dump of the virtual machine. Using the Libvirt, a result of the dump the memory 1GB of virtual machine, it took 4,189 ms.
- **GCORE**: GCORE [8] is a process debugging utility. GCORE allows to perform a memory dump of the virtual machine. Using the GCORE, a result of the dump the memory 1GB of virtual machine, it took 2,111 ms.
- **GDB**: GDB [9] (GNU Debugger) is to be able to debug the process under which the process is running. GDB allows to acquire memory snapshot by specifying the memory area to be taken. Using the GDB, a result of the dump the memory 1GB, it took 5,333 ms.

2.2 Memory Analysis

- **Memory Introspection**: LibVMI is a C library with Python bindings that makes it easy to monitor the low-level details of a running virtual machine by viewing its memory, trapping on hardware events, and accessing the vCPU registers. One of the most useful memory analysis tools is DRAKVUF with VMI. DRAKVUF [10] is one of the dynamic memory analysis tools. It utilizes VMI and volatility [11] to analyze guest OS memory in real time. It traces kernel functions instead of API calls and extracts deleted file information and registry.
- **Dynamic Malware Analysis**: Cuckoo Sandbox is the virtualization-based [12] analysis tool that

executes malware on guest OSes and provides a comprehensive analysis of the results. Cuckoo Sandbox has following features: extracting API calls, dumping memory of malware process, dumping whole memory and reporting analysis results. CW Sandbox is malicious code dynamic analysis tool that reports result of analyzed malware by uploading at the web page. It extracts the information of malware API calls such as Cuckoo Sandbox does. But, 'API trigger-based dump' does not provide.

- **Event-Driven Memory Dump**: Tomer Teller suggested it to handle memory dump timing. It compared previous dump policies such as interval-based and instruction-level dump. But, they can miss important information or cause large overhead. Therefore, he suggested a practical memory dump policy 'API trigger-based dump' by modifying cuckoo-mon.dll, monitoring module in Cuckoo Sandbox.

3. Design and Implementation of Mem-Shot

3.1 Memory Dump Acceleration

Analysis of memory dump operation: Libvirt is a library tool providing APIs for managing virtual machines. For instance, when user calls an API to acquire a memory dump from a certain virtual machine, Libvirt agent on the virtual machine saves the virtual machine memory inside of QEMU as a file. In this study, we analysis the process of the dump operation by using Perf which is a system performance analysis tool, included in the Linux kernel as shown in Table 1. As a result of analyzing the process to dump the inside memory of Libvirt, using report function of perf and arithmetic process operated for memory dump in QEMU, *copy_page_rep*() function induced the delay most except *qemu-system-x86_64* for virtual machine operation.

In an effort to minimize the computation overhead from *copy_page_rep*(), We took a look how the operation performs from memory dump request to QEMU in Libvirt. If user make a request for memory dump of the virtual machine in Libvirt, Libvirt performs following five steps operations as shown in Fig. 2: creating a ELF header (marked as ①) page table notes, the page table note, header create, creating ELF. Following that, *copy_ page_ rep*() extracts the information on the page table of the virtual machine from the page frame of the host machine's physical memory(marked as ②), and copies the information of the memory that virtual machine is using(marked as ③). It saves data in buffer area to save the copied memory information as disk and if saved buffer gets full, output the memory information to disk(marked as ④, ⑤). By analyzing the internal operation of Libvirt and QEMU, we could confirm delay of memory dump time comes from the speed-gap between memory and disk operations, which can be mapped into the operation marked as ⑤.

Table 1. Analysis of computation overhead of a memory dump operation

Overhead	Symbol
34.97 %	[*] qemu-system-x86_64 0xb2a6
11.15 %	[k] copy_page_rep
4.21 %	[k] intel_idle
2.77 %	[*] 0x2e87c
2.25 %	[*] 0x1332de
1.37 %	[*] 0x58bf
1.20 %	[*] vfprintf
1.05 %	[k] copy_user_enhanced_fast_string
0.98 %	[*] 0x8c53e

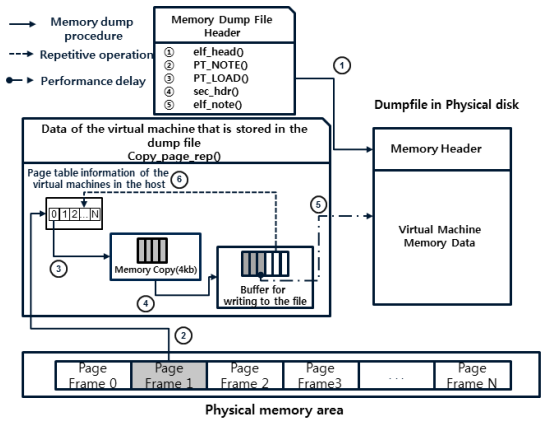

Fig. 2 Memory Dump Process of Libvirt

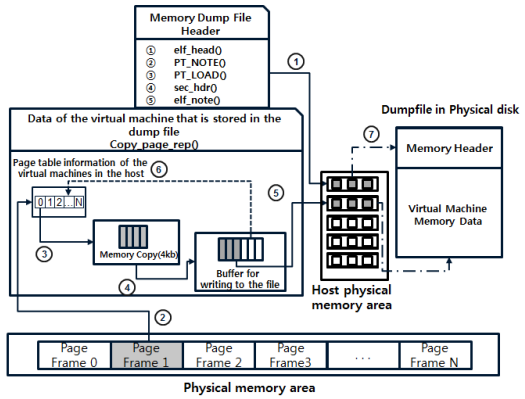

Fig. 3 Memory Dump Accelerations processing

- **In-Memory-based Memory Dump Acceleration**: As we described the dump operation overhead analysis, the dominant factor of overhead is from the relatively slow operation of disk in comparison to memory operation. As a remedy to this problem, we devised a new transaction mechanism to store the memory dump data by translating overall disk operation into memory operation.

As shown in Fig. 3, the recording memory dump schemes of Libvirt uses the page *copy_page_rep*() in QEMU. In the operation, the memory data used by the virtual machine is transferred to the disk of the host operating system. During this process, when the write operation of memory data into disk occurs, execution delay memory dump will occur. To minimize such a delay, we devised a new recording scheme by translating the operations (marked as ① and ⑤ in Fig. 2) into a pure memory operations (marked as ①, ⑤, and ⑦). More specifically, in an effort to remove the disk operation, we create a virtual memory space between memory and disk as described in Fig. 3.

- **API Trigger-based Memory Dump**: In this work, we incorporated a conventional Cuckoo Sandbox into our system as an underlying memory dump framework to extract arguments and addresses of APIs. In conventional Cuckoo Sandbox, it cannot take a memory snapshot at a time of a certain API called. To solve this problem, we focus on the fact that all Windows APIs write parameters on memory before the function is executed. Based on our above analysis, our goal is to implement API trigger-based memory dump technology, which dumps the memory at every API call to extract arguments and addresses.

To address problems, when API is triggered, API information should be bypassed to host OS in real time. In existing Cuckoo Sandbox case, user gets the information about API trace in a log format but it is not processed in real-time. According to our analysis on Cuckoo Sandbox operations, it extracts API call tracing by using dll-injection function-hooking [14]. To transfer API tracing logs, Cuckoo Sandbox utilizes TCP/IP protocols. Although TCP/IP communication delay is small, it is enough delay to execute multiple instructions which can cause hide code and data information before memory dump occurs. Therefore, it is inevitable to revise Cuckoo Sandbox architecture

to remove the gap between API call and memory dump. In addition, there are several APIs on which malware analysts focus according to the situation. For their convenience, we also need to provide interfaces which are enable malware analysts to select target APIs.

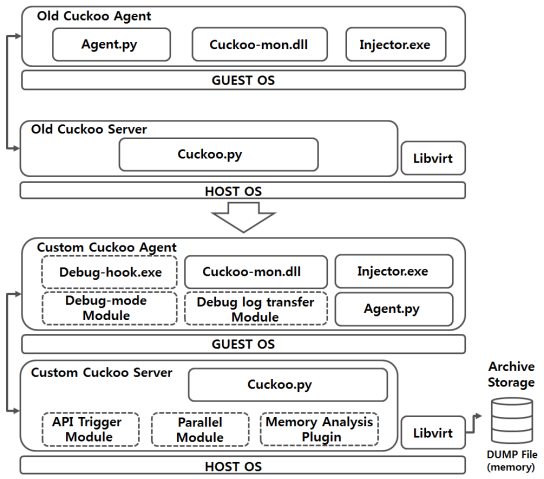

Fig. 4 Modified Cuckoo Sandbox for Event Driven Memory Dump

- **The delay problem of API trigger-based memory dump performance**: Cuckoo Sandbox use dll injection for hooking API. This method allows Cuckoo Sandbox to track the process flow, but it is impossible to stop the process in a specific time when API is called. This cannot prevent malware from hiding malicious codes and data again. Therefore, we revise hooking method in existing Cuckoo Sandbox and suggest API trigger-based memory dump technology.

To solve this issue, we use debug hooking [15] that allows debug hooking program to attach executing malware as shown Fig. 4. Hence, we can control malware execution in debug mode. If malware analysts want Cuckoo Sandbox to analyze malware, Cuckoo Sandbox executes malware and attaches to debug mode. When malware calls a specific API, malware is forced to be stopped by debugging operations and API call logs are sent to host OS. Host OS receives this information and dumps guest OS memory. After memory dump, malware resumes. Consequently, we can remove the time gap between malware process and memory dump.

Table 2. Source Code For Debug-hook-based Event Triggering

```
void DebugLoop()
{
   DEBUG_EVENT de;
   DWORD dwContinueStatus;
   while (WaitForDebugEvent(&de, INFINITE))
   {
      dwContinueStatus = DBG_CONTINUE;
      if(CREATE_PROCESS_DEBUG_EVENT ==  de.dwDebugEventCode)
      {
         OnCreateProcessDebugEvent(&de);
      }
      else if (EXCEPTION_DEBUG_EVENT ==   de.dwDebugEventCode)
      {
         if (OnExceptionDebugEvent(&de)) continue;
      }
      else if (EXIT_PROCESS_DEBUG_EVENT ==   de.dwDebugEventCode)
      {
         Break;
      }
      ContinueDebugEvent(de.dwProcessId,
       de.dwThreadId, dwContinueStatus);
    }
}
```

- **Debug-hook**: In order to solve problems with dll-injection function-hooking, we developed debugging based hooking techniques. Debugging

based hooking is a type of hooking technique which debugs the target process while debugging it. To be specific, we use debugging API functions to insert hooking functions after we make the target process paused. Consequently, we can control the target process interactively. In order for malicious processes to operate in debugging mode, we use EXCEPTION_DEBUG_EVENT and EXCEPTION_BREAKPOINT. We describe the example code in Table 2. Because our focus is to remove time gap which occurs in memory dump, we use our debugging based hooking technique as follows. First, we execute malware and make it a debuggee by attaching a debugger to malware. When malware calls a hooked API, we change the first instruction of such hooked API into INT (0xCC) and cause a breakpoint. After hooked API logs are transferred to Cuckoo Sandbox, we dump our target guest OS memory if hooked APIs belong to the trigger list. After this procedure is finished, we restore the first instruction of the hooked API for malware to execute its original instructions. Because we prevent malware from executing further instructions using breakpoint exceptions, we can dump memory at an exact time.

4. Performance Evaluation

In this paper, we evaluate the performance on three physical machines. Our experimental environment consists of three servers, CPU Xeon E5-2609(2.5GHz). One server computer which has 32GB RAM is responsible to control other nodes. The others which have 16GB RAM are in charge of analyzing malwares. We use Ubuntu 14.04 in host OS and Windows 7 64bit, and assign 1GB RAM to each virtual machine.

In order to compare the dump performance of existing Cuckoo Sandbox with revised Cuckoo Sandbox, a total of 2 details were compared.

4.1 Performance of Memory dump

This paper in the proposed implementation techniques in order to measure the performance of memory-based dump and acceleration of existing four dump tools were compared (GCORE, LibVMI, Libvirt, GDB) a. The experiment was measured the time to the target virtual machine that has been assigned the 1GB of memory to perform a memory dump.

The results of the experiment, the memory dump using GCORE of existing four types of dump tool showed the best performance. Using GCORE, it took 2,111ms when performing a memory dump of the virtual machine. GCORE as Linux basic tools of the dumping process, the host operating system, because it runs the virtual machine as a single process may utilize a GCORE, acquires memory information of the virtual machine. However, in the case of GCORE, but dump the speed of the process is fast, because they do not support the GCORE format in memory analysis tools, such as Volatility [11] and Rekall [16], memory analysis using the conventional memory analysis tools there is a drawback.

When applying the proposed memory-based memory dump scheme in this paper, it took 1,219ms when 1GB of memory dump. Existing memory dump tool in comparison with the memory dump the execution time performance ratio using LibVMI is improved 12,513 times, showed a performance improvement of 3.4 times the conventional Libvirt ratio, other GDB 9.2 times, GCORE ratio of 3.5 times the performance it shown in Fig. 5.

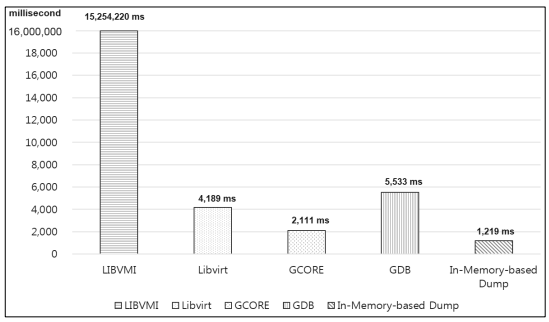

Fig. 5 Memory Dump Performance comparison

4.2 Measurement of Semantic gap

We tested the new function of API trigger applied to existing Cuckoo Sandbox and revised Cuckoo Sandbox. The method for the experiments are as follows. When malware calls a specific API, host OS dumps these memories in Guest OS. At this time, we compare methods of existing Cuckoo Sandbox using hooking with revised Cuckoo Sandbox using hooking based on debug mode. The Our sample malware writes tick count for every API and we compare this tick counts in existing Cuckoo Sandbox and revised Cuckoo Sandbox. With existing Cuckoo Sandbox, we compare the tick count for API calls with guest OS memory dump. The former has more 15 ticks than the latter has. In other words, dumping memory does not occur at the right time. On the other hand, in debug hooking mode case where we use revised Cuckoo Sandbox, we can't find the gap between the former and the latter. That means dumping memory occurs at the right time.

Table 3. Sample Malware Code

```
int main(void)
{
   DWORD startTickCount =   GetTickCount();
   DWORD currentTickCount;
   int client_socket;
   struct sockaddr_in server_addr;
   char obfus_ip[13]="@0w!!2@&^@";
   char dec_ip[13]="malw@re"
   bool checker=false;
   for ( ; ; )
   {
      currentTickCount = GetTickCount();
      if (currentTickCount - startTickCount   >= 10101)
      {
         break;
      }
      if (currentTickCount-startTickCount   >=5151)
      {
         if(!checker)
         {
            dec_ip=dec(obfus_ip);
            client_socket   = socket(PF_INET, SOCK_STREAM, 0);
            memset(dec_ip,0,strlen(dec_ip));
            checker=true;
         }
      }
    }
     return 0;
}
```

4.2 Experiment using a sample malware

For accurate verification, we created experiment sample malware as shown in Table 3. In this experiment, we tested both systems with malware which uses encrypted IP address. Before malware calls a certain API, it decrypts IP address and uses an input argument. Then, it re-encrypt IP address.

By using API call tracing, when socket functions were called in malware, dumping the memory occurs. We found IP address in the result from dumped memory, and could observe that IP address did not exist in dumped memory files in existing case. On the other hand, IP address existed, using debug hooking mode in revised case. It proves that API trigger-based memory dump technology can be used to facilitate other malware analysis.

5. Conclusion

In this paper, we developed API Trigger-based memory dump technology to extract hidden

information from sample-malware in memory. Existing Cuckoo Sandbox was modified to implement the API Trigger-based memory dump technology. We modify the hooking method that existing that Cuckoo Sandbox uses API triggering. With our system, there is no time gap between API calls and the time memory dump has been matched exactly. As a result, the temporarily raised plain text such as encrypted IP Address on memory can be accurately extracted. This experiment proves that we can detect hidden information before malware hides it again.

References

〔1〕 OU, I., AND YIM, K. Malware obfuscation techniques: A brief survey. In Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (Washington, DC, USA, 2010), BWCCA '10, IEEE Computer Society, pp. 297 - 300.

〔2〕 Themida : http:///www.oreans.com/themida.php

〔3〕 Tomer Teller, Enhancing Automated Malware Analysis Machines with Memory Analysis Blackhat 2014.

〔4〕 Claudio Guarnieri, "The Cuckoo Sandbox", http://http://www.cuckooSandbox.org.

〔5〕 Sang-Hoon Choi, Yu-Seong Kim, Ki-Woong Park, Toward Semantic Gap-less Memory Dump for Malware Analysis, ICNGC Conference, Jenuary 2016.

〔6〕 Haiquan Xiong et al. 'Libvmi: A Library for Bridging the Semantic Gap between Guest OS and VMM'. In: (2012), pp. 549 - 556.

〔7〕 M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann. Non-intrusive virtualization management using libvirt. In DATE '10, 2010

〔8〕 GCORE: http://man7.org/linux/man-pages/man1/gcore.1.html

〔9〕 GDB: https://www.gnu.org/software/gdb/

〔10〕 Tamas K. Lengyel, Steve Maresca, Bryan D. Payne, George D. Webster, Sebastian Vogl, and Aggelos Kiayias. Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system. In Proceedings of the 30th Annual Computer Security Applications Conference, December 2014.

〔11〕 Volatility :https://code.google.com/p/volatility/

〔12〕 Uhlig, R., Neiger, G., Rodges, D., Santoni, A. L., Martins, F. C. M., anderson, A. V., Bennett, S. M., Kagi A., Leung, F. H., and smith, L. Intel virtualization technology. Computer 38, 5 (2005), 48 - 56.

〔13〕 Fabrice Bellard. Qemu, a fast and portable dynamic translator. In Proceedings of the USENIX 2005 Annual Technical Conference, FREENIX Track, pages 41 - 46, 2005.

〔14〕 Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. KVM: the linux virtual machine monitor. In Ottawa Linux Symposium (July 2007), pp. 225 - 230 S. Fewer. Reflective DLL Injection. Technical report, Harmony Security, Oct. 2008.

〔15〕 Debug Hooking: https://msdn.microsoft.com/en-us/library/z2zscsc2.aspx

〔16〕 Rekall: http://www.rekall-forensic.com/

Author Introduction

◆ 최상훈

- 2014년 대전대학교 정보보안학과 학사
- 2016년 대전대학교 전산정보보안학과 석사
- 2016년~현재 공주대학교 산학협력단 연구원
- 관심분야: 클라우드 컴퓨팅 보안, 시스템 보안

◆ 박기웅

- 2005년 연세대학교 Computer Science 학사
- 2007년 KAIST Electrical Engineering 석사
- 2012년 KAIST Electrical Engineering 박사
- 2008년 Microsoft Research Asia, Wireless and Networking Grourp, Research Intern
- 2009년 Microsoft Research Redmond, Network Research Group, Research Intern Fellow
- 2012년 국가보안기술연구소 연구원
- 2012년~2016년 대전대학교 정보보안학과 교수
- 2016년~현재 세종대학교 정보보호학과 교수
- 관심분야: 시스템 보안, 모바일-클라우드 컴퓨팅, 보안 프로토콜, 디지털 포렌식 등