Toward Semantic Gap-less Memory Dump for Malware Analysis

Sang-Hoon Choi
System Security Lab.
Daejeon University
csh0052@gmail.com

Yu-Seong Kim
System Security Lab.
Daejeon University
kys10044@gmail.com

Ki-Woong Park*
Dept. of Information Security
Daejeon University
woongbak@dju.kr

***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. Our system makes it possible to take a memory snapshot at a time of a certain API called. 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.**

Keywords: Malware Analysis, Memory Dump, Hypervisor

I. 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, respectively. For example, instruction-level memory dump causes user-obstructive latency whenever a certain instruction is executed. Furthermore, malware authors use code obfuscation technique such as Themida [2] or put the garbage code which confuses malware analysts. On the other hand, periodical memory dump, it causes low overhead, but it is possible to miss important malicious code and data because code obfuscation techniques can hide them after execution.

As a remedy to the problems, Tomer Teller [3] has announced API-triggering memory dump analysis scheme to extract unpacked code and API parameters on which users want to focus, based on Cuckoo sandbox [4]. Although Tomer Teller's approach is effective, its modification is limited to cuckoo monitor *dll*. Consequently 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 customizations of the Cuckoo sandbox.

As a preliminary experiment, we performed memory dump in response to a certain API call. We found that the time gap exist between the time of the API call and the time of the memory dump time as shown Fig. 1. For example, API-triggering event occurs, 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 exact times, we used an automated malware analysis tool and revised the memory dump structure by analyzing and profiling the existing Cuckoo Sandbox and removed the time difference between the API call and dump. As a result, we can extract the string to be decoded only at specific times. In addition, we also consider feedback of malware analysts. From their opinions, they need to focus on some APIs rather than all APIs according to the situation. Therefore, we developed the system which removes API and dump time gaps and API selection interfaces.

This paper is organized as follows. In section 2, we will introduce the related work that performs API trigger-based memory dump technology, and in section 3 we will present our research motivation and show the problems by analyzing and

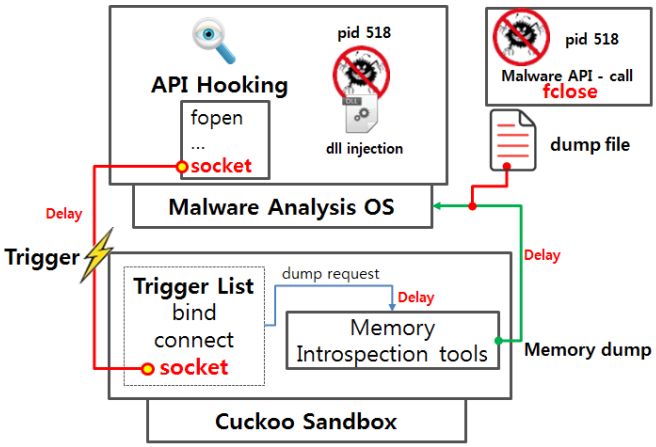

FIG 1. API TRIGGER-BASED MEMORY DUMP ON CUCKOO SANDBOX

*: Corresponding author

This work was supported by the Human Resource Training Program for Regional Innovation and Creativity through the Ministry of Education and National Research Foundation of Korea (2015H1C1A1035859).

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. Finally, in section 6, we describe our conclusions and future researches.

II. Related Work

A. Memory Introspection

LibVMI [5] 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 [6] with VMI.

DRAKVUF [6] is one of the dynamic memory analysis tools. It utilizes VMI [5] and volatility [7] to analyze guest OS memory in real time. It traces kernel functions instead of API calls and extracts deleted file information and registry. DRAKVUF is an excellent tool in real-time memory analysis. But, 'API triggering-based dump' is not provided.

B. Dynamic Malware Analysis

Cuckoo Sandbox is the virtualization-based [8] 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 triggering-based dump' does not provide.

C. 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 triggering-based dump' by modifying Cuckoo monitor *cuckoo-mon.dll* implemented on Cuckoo Sandbox. However, *dll* source code does not open to the public. To verify dump timing, we performed experiment sample malware.

III. Design & Implementation

In this work, we incorporated a conventional Cuckoo sandbox into our system as an underlying memory dump framework in an attempt 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 [9]. 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.

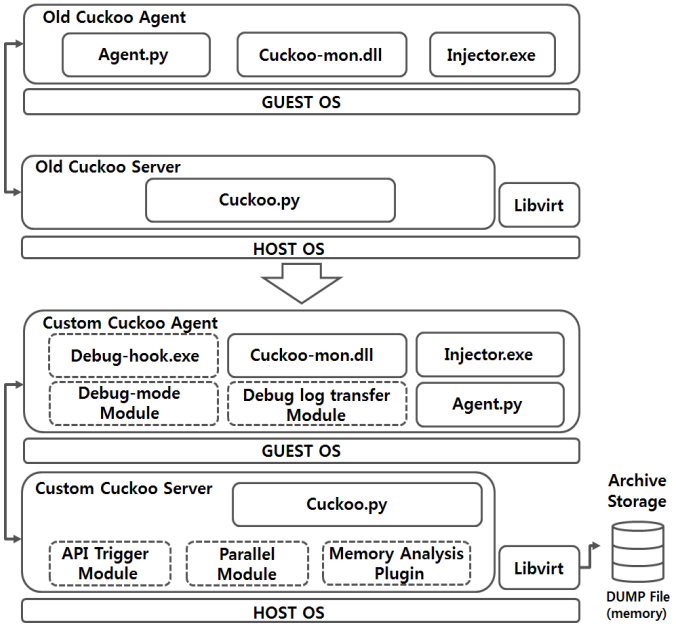

FIG 2. MODIFIED CUCKOO SANDBOX FOR EVENT DRIVEN MEMORY DUMP

A. The delay problem of API triggering-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 [10] that allows debug hooking program to attach executing malware as shown Fig2. 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 I. 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);
   }
}
```

B. 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 I. 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.

C. API Selection

With existing Cuckoo Sandbox approach, all APIs are traced every time. But, it can be too large amount of information. So, we need to select target APIs. To solve this problem, we provide configuration files which the lists of API trigger users want are. API calls matches listed APIs, host OS dumped guest OS memory by using Libvirt [11] in QEMU [12] library tools.

D. Memory Comparison Plug-in

Existing Cuckoo Sandbox contains a memory plug-in to compare two memory dump files to detect information changes. But, this plug-in cannot compare multiple memory files. However, it is necessary to compare multiple memory dumps to extract strings or parameters. Since latest malware can hide such information again, it is impossible to detect them with only two memory dump files. To solve this problem, our modified Cuckoo Sandbox can extract changed memory contents from all memory dump files. Users with the added plug-in can easily identify the changes in the memory information of the memory dump file in accordance with the API Trigger.

E. Parallel analysis

Cuckoo Sandbox can only analyze malware on a sing physical machine. In other words, It is impossible to analyze malware practically. While modified Cuckoo Sandbox creates a node for the analysis of a control node, it is possible at the same time to analyze a large number of malware based on the API Trigger memory dump.

IV. PERFORMANCE

In this paper, we evaluate the performance on three physical machines. Our experimental environment consist of three servers, CPU Xenon 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 XP SP2 – 32 bit, and assign 1GB RAM to each VM.

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

A. Mesaurement 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 II. MALWARE CODE SAMPLE

```
int main(void)
{
   DWORD startTickCount = GetTickCount();
   DWORD currentTickCount;
   int   client_socket;
   struct sockaddr_in   server_addr;
   char obfus_ip[13]="@@w!!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;
}
```

B. *Experiment using a sample malware*

We created experiment sample malware as shown in Table II. 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.

V. 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. As a further work, we will offer our malware analysis solution to malware analysts to develop malware code language for analysis.

REFERENCES

[1] YOU, 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] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In Proceedings of Network and Distributed Systems Security Symposium (NDSS'03), February 2003.

[6] 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.

[7] Volatility : https://code.google.com/p/volatility/

[8] 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.

[9] 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.

[10] Debug Hooking: https://msdn.microsoft.com/en-us/library/z2zscsc2.aspx

[11] M. Bolte, M. Sievers, G. Birkenheuer, O. Nieh¨orster, and A. Brinkmann. Non-intrusive virtualization management using libvirt. In DATE '10, 2010.

[12] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In Proceedings of the USENIX 2005 Annual Technical Conference, FREENIX Track, pages 41–46, 2005.