Alkanet: A Dynamic Malware Analyzer based on Virtual Machine Monitor

Yuto Otsuki, Eiji Takimoto, Takehiro Kashiyama, Shoichi Saito, Eric W. Cooper, and Koichi Mouri

***Abstract*—Recently, malware has become a major security threat to computers. Responding to threats from malware requires malware analysis and understanding malware behavior. However, malware analyst cannot spend the time required to analyze each instance of malware because unique variants of malware emerge by the thousands every day. Dynamic analysis is effective for understanding malware behavior within a short time. The method of analysis to execute the malware and observe its behavior using debugging and monitoring tools. We are developing Alkanet, a malware analyzer that uses a virtual machine monitor based on BitVisor. Alkanet can analyze malware even if the malware applies anti-debugging techniques to thwart analysis by dynamic analysis tools. In addition, analysis overhead is reduced. Alkanet executes malware on Windows XP, and traces system calls invoked by threads. Therefore, the system can analyze malware that infects other running processes. Also, the system call trace logs are obtained in real time via a IEEE 1394 interface. Other programs can readily examine the log and process the analysis results to understand intentions of malware behavior. In this paper, we describe the design and implementation of Alkanet. We confirm that Alkanet analyzes malware behaviors, such as copying itself, deleting itself, and creating new processes. We also confirm that Alkanet accurately traces threads injected by malware into other processes.**

***Index Terms*—malware analysis, dynamic analysis, virtual machine monitor, system call tracing.**

I. Introduction

RECENTLY, malware has become a major security threat on computers. According to a report released by Symantec Corporation, more than 403 million unique variants of malware were detected in 2011 [1]. This number was an increase of 41% over the previous year. Responding to this threat requires analysis and understanding of malware behavior. However, malware analysis cannot spend a lot of time on each malware because new variants emerge by the thousands every day.

The first step of malware analysis is a dynamic analysis to gain a understanding of the malware's general behavior. In this step, analysts execute the malware and observe its behavior using debuggers and monitoring tools. The next step targets only complicated or notable malware. In such cases, analysts do a detailed dynamic analysis or static analysis using disassemblers and debuggers to read the malware code. Malware analysts need to form a summary of malware in a short time because new malware is constantly emerging. Therefore, we focus attention on the first dynamic analysis.

Dynamic analysis differs from static analysis in that it is not thwarted by techniques to interfere with program analysis such as packing and obfuscation. So dynamic analysis can provide summary reports of malware in a short time. However, recent malware has applied anti-debugging techniques [2], [3]. When malware detect that it has been analyzed by dynamic analysis tools, it may attempt to evade analysis or tamper with the analysis tools. Traditional dynamic analysis tools run using services provided by Windows to assist in debugging. However, these tools and services do not run stealthily, hidden from the program being debugged, because their purpose is to debug legitimate software. Therefore, malware can detect them easily. To analyze malware that applies anti-debugging techniques, these techniques need to be disabled one by one using a debugger or kernel-mode driver. Alternatively, analysts should use other analysis tools and not depend on general debugging assistant services. However, it is very difficult to evade all anti-debugging techniques. In addition, side effects to the environment in which the malware is executed are increased by falsifications needed to evade anti-debugging techniques, which cause increased overhead and inaccurate analysis.

Analysts and researchers need to use dynamic analysis to reduce side effects to environments for executing malware. Observing from outside the execution environment is an effective method of reducing the side effects. Some dynamic analysis system implementations have been based on virtual machine monitors (a.k.a. VMM) or emulators. VMMs and emulators run under a higher privilege level than the operating systems in virtual machines, which can observe all user-mode processes and the operating system in a virtual machine transparently. However, emulating the whole environment by software alone incurs a huge processing overhead. In addition, malware can detect general VMM easily. The reason is that general VMM emulates specific hardware and has communication interfaces with the guest operating system.

Consideration of the targets of analysis reveals another problem. Recently, more and more malware has the capability of spreading outside the range of a single process. The purposes of such spreading behavior is to conceal activities and to make analysis difficult. Sophisticated malware injects malicious codes and threads into other processes. Existing dynamic analysis systems cannot trace malicious threads injected into legitimate processes because these systems distinguish the current executing task by process level.

It is also necessary to give consideration to granularity of analysis. Instruction level analysis can analyze particularly. However, it causes too much overhead and it is hard to interpret the intentions of malware behavior. An API trace that records Windows API functions called by malware gives

Manuscript received July 23, 2012; revised August 10, 2012.

Y. Otsuki is with Graduate School of Science and Engineering, Ritsumeikan University 1-1-1 Nojihigashi, Kusatsu, Shiga 525-8577 Japan e-mail: yotuki@asl.cs.ritsumei.ac.jp.

T. Kashiyama is with Ritsumeikan Global Innovation Research Organization, Ritsumeikan University.

S. Saito is with Graduate School of Engineering, Nagoya Institute of Technology.

E. Takimoto, E.W. Cooper and K. Mouri are with College of Information Science and Engineering, Ritsumeikan University.

abstract and useful information, and can be obtained in a short time. There is the possibility that the malware alters user-mode memory space to evade tracing. However, malware needs to invoke system calls to affect to the environment when it is running on user-mode. Therefore, system call tracing is an effective method for malware analysis.

We are developing the system a dynamic analysis system to meet the following requirements, based on the above-mentioned reasons.

1) The system can analyze malware that applies anti-debugging techniques.
2) The system can analyze malware that infects other running processes.
3) The system traces system calls invoked by malware.
4) The analysis method is one that mitigate analysis overhead.

To meet these requirements, we are developing Alkanet, a malware analyzer using a VMM. The remainder of this paper is organized as follows. Section II gives an overview of Alkanet. Section III details our system call tracing method. Section IV presents malware analysis results and other considerations of using Alkanet. Section V describes our future works. Section VI summarizes the related works. Finally, Section VII gives the conclusions of this paper.

II. Overview of Alkanet

A. *Outline*

Alkanet is a dynamic analysis system for malware analysis using VMM. Malware analyzers implemented in VMM can analyze with a higher privilege level than malware. So many anti-debugging techniques would be ineffective for Alkanet. Alkanet can observe running malware without the malware interfering.

From two viewpoints, API level tracing is better suited for analyzing malware within a short time than instruction level analysis. One viewpoint is the ease of understanding intention of a behavior. The other is analysis overhead. In addition, malware running in user mode needs to invoke system calls to affect the environment. Therefore, Alkanet traces system calls invoked by malware and analyzes malware behaviors.

Achievement of system call tracing requires hooking every system call invoked by malware and getting the arguments and return value. In addition, the tracing requires analysis of the meaning of the arguments and return value to get detailed information of system calls invoked. However, a VMM cannot get abstract information on the operating system level. A VMM cannot call the API provided by operating system. The reason is that the VMM runs outside of the guest operating system. Many existing dynamic analysis systems using a VMM or emulator resolve this problem by receiving internal information from their agent processes or drivers running in Windows. However, this solution carries the risk that malware detects or tampers with the agent and the communication interfaces. A VMM or emulator can conceal or protect the agent and the communication interfaces from malware. These efforts cause an increase in analysis overhead. Our approach to the problem is that Alkanet itself refers to a memory region of Windows and obtains detailed information of Windows. VMM can access all memory regions of the virtual machine containing the operating system because VMM runs at a higher privilege level than the guest operating system.

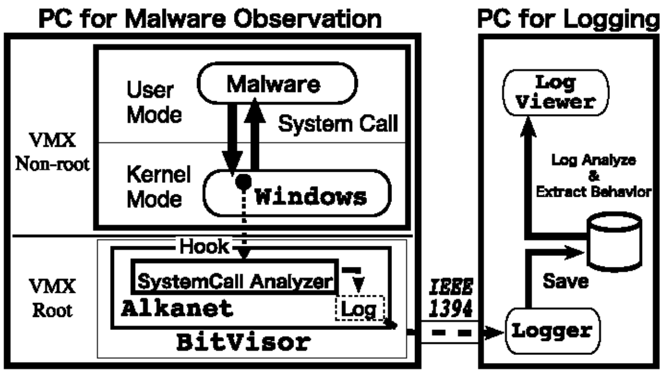

Fig. 1. Construction of Alkanet

Malware behaviors can be analyzed from the system call trace logs. However, the system call tracer in Alkanet records a high volume of logs in cases where the target of analysis is large-scale or sophisticated malware. For these reasons, our system extracts a summary of malware behavior from further analysis of the system call trace logs.

B. *Construction*

Figure 1 presents the construction of Alkanet. Alkanet is implemented based on BitVisor [4]. BitVisor runs directly on the hardware and does not require a host operating system, and instead runs on processors with Intel Virtualization Technology (a.k.a. Intel VT). Intel VT assists virtualization by VMM. Therefore, BitVisor runs faster than emulators and VMMs implemented in software only. BitVisor can run Windows without requirement of modifications. In addition, BitVisor adopts the parapass-through architecture and does not emulate specific hardware. BitVisor provides the physical hardware for a guest operating system. Therefore, malware cannot detect BitVisor by characteristics of hardware, unlike emulators and VMMs that emulate specific hardware. Furthermore, Alkanet adopts Windows XP 32bit edition as its guest operating system. Alkanet executes malware in this environment. Alkanet hooks invoked system calls in this environment and records the number of system calls, their arguments, return values, and so on.

Another machine obtains the system call trace logs via IEEE 1394 interface. IEEE 1394 has direct read and write access to the physical memory of connected devices. This direct access allows the tracing logs to be obtained without the malware detecting or interfering. Other programs can readily examine the log and process the analysis results to understand intentions of malware behavior.

C. *Observed System Calls and Getting Information*

Table I gives an example of behaviors and system calls that Alkanet observes. Typical malware functions consist of these system calls. The specific operations observed are as follows.

- File operations: open, create, delete, read, and write
- Registry operations: open key, create key, query value, and set value
- Network operations: send and receive

TABLE I
SYSTEM CALLS OBSERVED BY ALKANET

Behaviors	Examples of System Calls
File operations	NtCreateFile, NtReadFile, NtWriteFile
Registry operations	NtCreateKey, NtQueryValueKey, NtSetValueKey
Network operations	NtDeviceIoControlFile, NtReadFile, NtWriteFile
Process creations and terminations	NtCreateProcessEx, NtTerminateProcess
Driver loads and unloads	NtLoadDriver, NtUnloadDriver
Code injection to other processes	NtCreateThread, NtWriteVirtualMemory

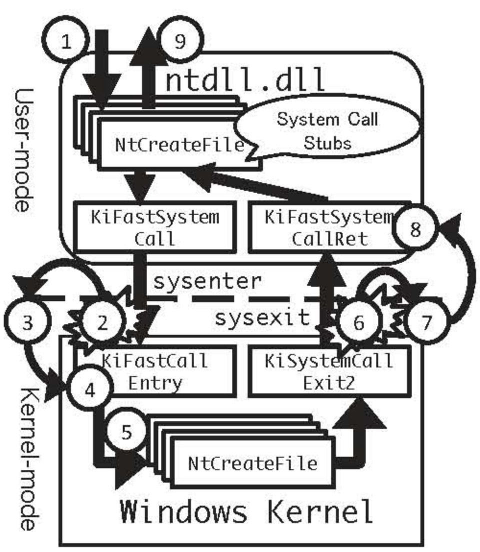

Fig. 2. Flow of System Call Hooking

- Process creations and terminations
- Driver loads and unloads
- Code injection to other processes

We have to distinguish a system call invoker by means of thread level because the execution unit in Windows is a thread. In addition, there are malicious threads in legitimate processes by code injection. Therefore, to distinguish a system call invoker, Alkanet obtains the Cid (pair that includes process id and thread id) and image name. In addition, it obtains system call arguments and return value in order to analyze malware behaviors. However, raw arguments and return value lack enough information for analysis because they consist of pointers and Windows-specific data structures. Therefore, the necessary information for analysis is obtained by interpreting these data structures. In the process described above, Alkanet obtains the following information.

- System call invoker's Cid and image name
- System call number
- System call arguments and return value
- Complementary information for Windows specific data structures

III. SYSTEM CALL TRACING

A. System Call Hooking

A system call in Windows XP 32bit edition usually uses the sysenter and sysexit instructions. Sysenter enters from user mode to kernel mode. Sysexit returns from kernel mode to user mode. To get inputs given to system calls and their results, Alkanet hooks both sysenter and sysexit.

Figure 2 presents the flow of system call hooking by Alkanet. The following steps detail the flow.

1) Malware invokes a system call.
2) At the entry point of kernel-mode, transition from Windows to Alkanet occurs by a breakpoint.
3) Alkanet gets the necessary information.
4) Alkanet returns control to Windows.
5) Windows executes kernel functions.
6) At the exit point of kernel-mode, transition from Windows to Alkanet occurs by breakpoint again.
7) Alkanet gets the necessary information containing system call results.
8) Alkanet returns control to Windows.
9) Windows returns control to malware.

Alkanet uses hardware breakpoints to hook system calls. Alkanet sets breakpoints on entry point of KiFastCallEntry and exit point of KiSystemCallExit2. KiFastCallEntry is sysenter destination. KiSystemCallExit2 contains sysexit. These symbols are public by Microsoft [5]. We can get address of these symbols. Some anti-debugging techniques applied by malware detect hardware breakpoints. Countering such techniques requires concealment of the hardware breakpoints used by Alkanet. A VMM running on Intel VT can catch events tine which a guest operating system has modified debug registers. So Alkanet can conceal hardware breakpoints with a low overhead.

A system call consists a pair of sysenter and sysexit calls. We need to associate these logs. However, this does not mean that these logs are always consecutive. Therefore, Alkanet hooks each point individually. In log analysis phase, our system associates these logs using information of the system call number and invoker.

B. Identifying Invoked System Call

When a system call is invoked, Windows sets the system call number in the EAX register. Alkanet can get the system call number from the EAX register when hooking sysenter. On the other hand, the value of the EAX register has already been changed when hooking sysexit. System calls are invoked via stubs implemented in ntdll.dll. These stubs each have the same name symbol as their library function. For example, NtCreateFile system call is invoked via NtCreateFile function of ntdll.dll. Therefore, Alkanet can identify invoked system call by the return address pushed on the stack.

C. Identifying Invoker Process and Thread

When Alkanet hooks a system call invoked on Windows, Alkanet gets information of the process that invoked the system call. Windows has data structures of each processor

```
NTSTATUS NtCreateFile(
  __out      PHANDLE FileHandle,
  __in       ACCESS_MASK DesiredAccess,
  __in       POBJECT_ATTRIBUTES ObjectAttributes,
  __out      PIO_STATUS_BLOCK IoStatusBlock,
  __in_opt   PLARGE_INTEGER AllocationSize,
  __in       ULONG FileAttributes,
  __in       ULONG ShareAccess,
  __in       ULONG CreateDisposition,
  __in       ULONG CreateOptions,
  __in       PVOID EaBuffer,
  __in       ULONG EaLength
);
```

Fig. 3. Declaration of NtCreateFile [6]

state, called Processor Control Region (a.k.a. PCR) and Processor Control Block (a.k.a. PRCB). These data structures map on to the FS segment of each processor. The Windows kernel and hardware abstraction layer use these data structures. PCR and PRCB have the address of the thread object running currently. Therefore, Alkanet can get information of invoking process from the thread object referred by these data structures.

A thread object on Windows has pair of process id and thread id, called Cid. Each thread object has also pointer to the process object the thread belongs to. The process object has the image name of the process. Therefore, Alkanet can get the Cid and image name of the process from the thread object and the process object.

D. Getting Return Value and Arguments

Windows APIs store return value to the EAX register and store arguments to the stack. Alkanet gets the return value from the EAX register and gets arguments from the stack. System calls in Windows save the value of the ESP register at the time to the EDX register to give the top of the stack in user mode to the Windows kernel. The sysexit instruction loads the value of the ECX register into the ESP register. Alkanet gets the top of the stack in user-mode from the EDX register when hooking sysenter and gets it from the ECX register when hooking sysexit.

Raw arguments and return values lack information to analyze because they consist of pointers and Windows specific data structures. Alkanet supplies the required information to analysis by referring to the memory region of Windows and interpreting these data structures. For example, NtCreateFile is a system call to create or open a file. Figure 3 shows the NtCreateFile declaration. We need to understand which files malware opened or created. Thus, it is necessary to get the file path that is passed to NtCreateFile. The third argument of NtCreateFile is a pointer to an OBJECT_ATTRIBUTES structure. The OBJECT_ATTRIBUTES structure is used to set attributes of a Windows internal object. The structure contains the UNICODE_STRING type field, called ObjectName. In the case of NtCreateFile, the field is a unicode string for the file path. The third argument is read only, indicated by the argument annotation `__in` [7]. Therefore, Alkanet gets the arguments in both hooks.

Obtaining more detailed information about the the file requires referring to the corresponding file object. Windows manages resources (files, registries, processes, etc.) as objects. Each user-mode process has a handle table to manage objects opened by the process. A user-mode process interacts with resources using the handle corresponding to the object. In the case of NtCreateFile, the first argument is a pointer to a variable to receive a handle corresponding to the created or opened file object. The file object contains information about the file. For example, the object has a UNICODE_STRING type field that has the file path, called FileName. Alkanet refers to the memory region of Windows and gets detailed information of objects when the need arises. However, in the NtCreateFile case, the first argument is annotated with `__out`. `__out` arguments are written into by Windows kernel functions. Therefore, Alkanet gets these only when sysexit hooking.

IV. Evaluation

A. Analysis Target Samples and Evaluation Methods

To confirm that Alkanet is effective for real malware analysis, we analyzed real malware samples using Alkanet. The samples are actual instances of malware recorded in CCC DATAset 2011 [8]. Here, we call the samples SdBot.exe, Palevo.exe and Polipos.exe based on the names assigned by some anti-virus software.

We executed these malware samples, traced invoked system calls, and analyzed the logs. We checked the validity of our analysis results by comparison with reports on the malware by anti-virus vendors. In this regard, however, there are large number of variants for the each of the malware and the variants differ from each other in the details. The two malware samples are not necessarily the same instance even if they have been detected as the same name by anti-virus software. It is hard to fully match the malware behaviors actually observed to reports by anti-virus vendors. Therefore, we evaluated whether our system could observe malware behavior characteristics in common with reports from several anti-virus vendors. In addition, we also confirmed whether there are variants of the malware that exhibit minor behaviors observed by Alkanet.

In this evaluation, we did not connect our system to networks. The reason is that Alkanet does not filter any network activity in its current implementation. We must prevent the malware being analyzed from performing actual attacks against real computers and servers.

B. SdBot

Figure 4–7 presents a portion of the trace logs for SdBot.exe. The meaning of each item in the log entry is as follows.

No.	Consecutive number for this entry in the trace log
Time	CPU time when this entry have been recorded
Cid	Cid of system call invoker
Name	Image name of system call invoker
Type	Whether sysenter or sysexit entry
Ret	Return value (only sysexit entry)
SNo.	Number and name of invoked system call
Note	Additional information such as about arguments

SdBot.exe ran as a process with process id 158. The SdBot.exe process read file of itself (No. 1356, 1357), created a file named ssms.exe in `C:\WINDOWS\System32` folder (No. 4796, 4800) and wrote the file (No. 4806, 4807). This

```
No. :    1356       Time:    1697432035
Cid :    158.544    Name:    SdBot.exe
Type:    sysenter
SNo.:    b7 (NtReadFile)
Note:    \...\My Documents\SdBot.exe

No. :    1357       Time:    1697432105
Cid :    158.544    Name:    SdBot.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    b7 (NtReadFile)
Note:    \...\My Documents\SdBot.exe

No. :    4796       Time:    1697997485
Cid :    158.544    Name:    SdBot.exe
Type:    sysenter
SNo.:    25 (NtCreateFile)
Note:    \??\C:\WINDOWS\system32\ssms.exe

No. :    4800        Time:    1697998049
Cid :    158.544     Name:    SdBot.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    25 (NtCreateFile)
Note:    \WINDOWS\system32\ssms.exe

No. :    4806        Time:    1697999212
Cid :    158.544     Name:    SdBot.exe
Type:    sysenter
SNo.:    112 (NtWriteFile)
Note:    \WINDOWS\system32\ssms.exe

No. :    4807        Time:    1697999802
Cid :    158.544     Name:    SdBot.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    112 (NtWriteFile)
Note:    \WINDOWS\system32\ssms.exe
```

Fig. 4. SdBot.exe: copying file of itself to system32 folder

```
No. :    8652        Time:    1700450001
Cid :    65c.2c0     Name:    ssms.exe
Type:    sysenter
SNo.:    e0 (NtSetInformationFile)
Note:    DELETE: \...\My Documents\SdBot.exe

No. :    8653        Time:    1700450168
Cid :    65c.2c0     Name:    ssms.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    e0 (NtSetInformationFile)
Note:    DELETE: \...\My Documents\SdBot.exe
```

Fig. 5. SdBot.exe: deleting file of itself

```
No. :    8656        Time:    1700451656
Cid :    65c.2c0     Name:    ssms.exe
Type:    sysenter
SNo.:    f7 (NtSetValueKey)
Note:    \REGISTRY\...\WINDOWS\CURRENTVERSION\RUN

No. :    8657        Time:    1700452279
Cid :    65c.2c0     Name:    ssms.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    f7 (NtSetValueKey)
Note:    \REGISTRY\...\WINDOWS\CURRENTVERSION\RUN
```

Fig. 6. SdBot.exe: setting to start automatically when system is rebooted

```
No. :    9390        Time:    1700652599
Cid :    65c.678     Name:    ssms.exe
Type:    sysenter
SNo.:    25 (NtCreateFile)
Note:    \??\SICE

No. :    9391        Time:    1700652627
Cid :    65c.678     Name:    ssms.exe
Type:    sysexit
Ret :    c0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
SNo.:    25 (NtCreateFile)
Note:    \??\SICE

No. :    9414        Time:    1700746353
Cid :    65c.678     Name:    ssms.exe
Type:    sysenter
SNo.:    25 (NtCreateFile)
Note:    \??\REGMON

No. :    9415        Time:    1700746367
Cid :    65c.678     Name:    ssms.exe
Type:    sysexit
Ret :    c0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
SNo.:    25 (NtCreateFile)
Note:    \??\REGMON

No. :    9416        Time:    1700761977
Cid :    65c.678     Name:    ssms.exe
Type:    sysenter
SNo.:    25 (NtCreateFile)
Note:    \??\FILEMON

No. :    9417        Time:    1700762003
Cid :    65c.678     Name:    ssms.exe
Type:    sysexit
Ret :    c0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
SNo.:    25 (NtCreateFile)
Note:    \??\FILEMON
```

Fig. 7. SdBot.exe: scanning device files used by dynamic analysis tools

behavior is presented by Figure 4. After the behavior, the SdBot.exe process executed ssms.exe. The ssms.exe process deleted the original file of SdBot.exe (Figure 5).

We confirmed that the ssms.exe process sets specific registry keys. Figure 6 shows that the ssms.exe process sets the Run key. Applications registered with the Run key start automatically when user logs onto the system. We confirmed that ssms.exe was registered with that key.

The ssms.exe process scanned device files regularly used by dynamic analysis tools. This behavior is one of general anti-debugging techniques. Figure 7 presents some of the recorded behaviors. Some dynamic analysis tools create specific device files. Therefore, malware can detect the tools by confirming the existence of specific device files. This kind of anti-debugging technique is ineffective for Alkanet because Alkanet does not create specific device files.

In addition, Alkanet confirmed the following malware behaviors.

- Executing cmd.exe and regedit.exe
- Creating and deleting temporary files
- Modifying settings of services and networks

Malware detected as SdBot by anti-virus software are backdoor Trojans [9], [10]. They await commands by attackers in internet relay chat (a.k.a. IRC). Additionally, there are many variants of SdBot. The infection method of SdBot is that the malware copies a file of itself to the Windows system folder. Each file copied then has a similar name to a regular Windows execution file. SdBot sets the copied file in Run key so that it will start automatically when the system has restarted.

In this analysis, we conclude that our system successfully analyzed the infection process of SdBot. It is because, as previously mentioned, our system could observe the behaviors

```
No. :    3945        Time:    161921464
Cid :    534.538     Name:    Palevo.exe
Type:    sysenter
SNo.:    30 (NtCreateProcessEx)
Note:    \...\My Documents\Palevo.exe

No. :    3946        Time:    161922097
Cid :    534.538     Name:    Palevo.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    30 (NtCreateProcessEx)
Note:    PID: 53c, ProcessName: Palevo.exe

No. :    3971        Time:    161926922
Cid :    534.538     Name:    Palevo.exe
Type:    sysenter
SNo.:    101 (NtTerminateProcess)
Note:    PID: 534, ProcessName: Palevo.exe

No. :    3972        Time:    161926935
Cid :    534.538     Name:    Palevo.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    101 (NtTerminateProcess)
Note:    PID: 534, ProcessName: Palevo.exe
```

Fig. 8. Palevo.exe: restarting itself

such as Sdbot.exe dropping ssms.exe in the system32 folder and setting it in the Run key. It coincides with characteristics of SdBot that name of the ssms.exe is similar name of a Windows regular process, called smss.exe (Session Manager Subsystem). In addition, we confirmed that SdBot.exe modified network settings. However, we could not analyze the behaviors using IRC in detail because our analysis system was not connected to the network.

C. Palevo

Figure 8–10 presents part of the trace logs for Palevo.exe. Palevo.exe ran as a process with process id 534. Figure 8 presents a behavior in which the malware restarted itself promptly. This behavior is an anti-debugging technique to evade attaching debugger to. General debugger cannot be attached to more than one process. Therefore, the malware tried restarting itself and leaving behind debugger. The technique is ineffective for Alkanet because Alkanet hooks all invoked system calls and traces processes spawned by malware.

Figure 9 presents a part of the behaviors by new Palevo.exe process with process id 53c. The new process read file itself (No. 3988, 3989) and wrote psyjo3.exe in recycle bin of non-existent user (No. 4128, 4129). The behavior is copying file of malware itself. The psyjo3.exe was registered to restart when system would be restarted.

In addition, we confirmed that Palevo.exe infected to other process. Figure 10 presents the behavior. Palevo.exe wrote to memory space of explorer.exe (No. 4158, 4159) and created a thread in the process (No. 4164, 4165). The purpose of the behavior is to conceal its malicious thread in explorer.exe and conceal its main threats. Palevo.exe process itself exited shortly after the behavior. We confirmed suspicious behaviors by the malicious thread in explorer.exe. For example, the malicious thread accessed network devices and set some registry values managing applications associated with file extensions.

Malware detected as Palevo by anti-virus software are bots forming Mariposa botnet [11]. The behavior characteristics

```
No. :    3988        Time:    161934677
Cid :    53c.540     Name:    Palevo.exe
Type:    sysenter
SNo.:    b7 (NtReadFile)
Note:    \...\My Documents\Palevo.exe

No. :    3989        Time:    161934724
Cid :    53c.540     Name:    Palevo.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    b7 (NtReadFile)
Note:    \...\My Documents\Palevo.exe

No. :    4128        Time:    161959968
Cid :    53c.540     Name:    Palevo.exe
Type:    sysenter
SNo.:    112 (NtWriteFile)
Note:    \RECYCLER\S-1-5-21-0243...\psyjo3.exe

No. :    4129        Time:    161960277
Cid :    53c.540     Name:    Palevo.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    112 (NtWriteFile)
Note:    \RECYCLER\S-1-5-21-0243...\psyjo3.exe
```

Fig. 9. Palevo.exe: copy file of itself to recycle bin of non-existent user

```
No. :    4158        Time:    161976728
Cid :    53c.540     Name:    Palevo.exe
Type:    sysenter
SNo.:    115 (NtWriteVirtualMemory)
Note:    PID: f8, ProcessName: explorer.exe

No. :    4159        Time:    161976811
Cid :    53c.540     Name:    Palevo.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    115 (NtWriteVirtualMemory)
Note:    PID: f8, ProcessName: explorer.exe

No. :    4164        Time:    161977822
Cid :    53c.540     Name:    Palevo.exe
Type:    sysenter
SNo.:    35 (NtCreateThread)
Note:    PID: f8, ProcessName: explorer.exe

No. :    4165        Time:    161977934
Cid :    53c.540     Name:    Palevo.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    35 (NtCreateThread)
Note:    Cid: f8.544, ProcessName: explorer.exe
```

Fig. 10. Palevo.exe: thread injection to explorer.exe

of Palevo contain dropping copies to user folders or system folders. Some of variants drop copies to recycle bins [12], [13]. The behavior characteristics also contain connecting to remote servers by its malicious thread concealed in explorer.exe [12], [14]. In addition, there are also variants containing the behaviors setting some registry values managing applications associated with file extensions [14]. These behaviors were observed in this our evaluation. Infection methods of Palevo are way to use peer-to-peer networks or removable medias. However, we could not confirmed the behaviors in detail. It is because our system did not be connected to networks and removable medias in this our evaluation.

D. Polipos

Figure 11, 12, 13 present a part of trace logs for Polipos.exe. Figure 14, 15 present a part of results of log analysis.

```
No. :    5786        Time:    677232506
Cid :    54c.6cc     Name:    Polipos.exe
Type:    sysenter
SNo.:    30 (NtCreateProcessEx)
Note:    \...\My Documents\Polipos.exe

No. :    5787        Time:    677233114
Cid :    54c.6cc     Name:    Polipos.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    30 (NtCreateProcessEx)
Note:    PID: bc, ProcessName: Polipos.exe
```

Fig. 11. Polipos: swanning process

```
No. :    6339        Time:    689820849
Cid :    bc.304      Name:    Polipos.exe
Type:    sysenter
SNo.:    35 (NtCreateThread)
Note:    PID: b0, ProcessName: explorer.exe

No. :    6340        Time:    689820959
Cid :    bc.304      Name:    Polipos.exe
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    35 (NtCreateThread)
Note:    Cid: b0.1e8, ProcessName: explorer.exe
```

Fig. 12. Polipos: thread injection to explorer.exe

Polipos.exe process ran as a process with process id 54c. Figure 11 presents that the Polipos.exe process has spawned new Polipos.exe process using NtCreateProcessEx. The new Polipos.exe process ran as a process with process id bc.

Figure 12 presents that Polipos.exe process with process id bc injected a thread to explorer.exe process using NtCreateThread. The behavior is a kind of code injections. The malicious thread in explorer.exe injected then ran with thread id 1e8. The thread copied file of Polipos.exe, and tried to connect networks.

Figure 13 present that Polipos.exe process tried to inject a thread to System process using NtCreateThread. However, this NtCreateThread was failed because return value was STATUS_INVALID_HANDLE.

Polipos.exe created threads into some other processes such as svchost.exe, service.exe, winlogon.exe, alg.exe, rundll32.exe, sqlservr.exe and lsass.exe. Figure 14 shows thread tree. It generated by analyzing system call logs and also by tracing threads that derived from other injected threads. On the figure, the Polipos's thread (Cid: 54c.1c8) created new thread (Cid: 480.2c4) into svchost.exe (process id: 480). This thread (Cid: 480.2c4) created a new thread (Cid: 480.22c). This thread (Cid: 480.22c) created many new threads whose Cids are 480.38c, 480.360, 480.720, and 480.24c.

According to the thread (Cid:480.720), code injection to rundl32.exe was found. It has created new thread (Cid: 220.7f8). Alkanet could trace other derived threads from the thread (Cid: 220.7f8). Similarly, we found that the malicious thread on svchost.exe (Cid: 480.22c) also injected threads into other processes, such as explorer.exe and alg.exe. In this way, Alkanet can trace threads injected to other processes and derived from it also.

We focused attention on a Cid 480.41c thread. Figure 15 presents that the thread tried to open files such as drwebase.vdb, avg.avi, vs.vsn, anti-vir.dat. We recognized

```
No. :    6383        Time:    691816271
Cid :    bc.304      Name:    Polipos.exe
Type:    sysenter
Ret :    - (-)
SNo.:    35 (NtCreateThread)
Note:    PID: 4, ProcessName: System

No. :    6384        Time:    691816285
Cid :    bc.304      Name:    Polipos.exe
Type:    sysexit
Ret :    c0000008 (STATUS_INVALID_HANDLE)
SNo.:    35 (NtCreateThread)
Note:    PID: 4, ProcessName: System
```

Fig. 13. Polipos: thread injection to System

from return values that there were not these files. In addition, other malicious threads tried to open these files. These files are used by some anti-virus software. Therefore, the behavior is a part of behavior deleting these files to prevent the malware itself being detected by anti-virus software.

Malware detected as Polipos or Polip by anti-virus software infect running processes [15], [16]. The malware infect most of running processes and conceal existence of them. In addition, the malware also have a behavior deleting specific files of anti-virus software. We could confirmed these behaviors from analysis results by Alkanet as previously mentioned. Infection methods of Polipos are way to use Gnutella peer-to-peer networks. However, we could not confirmed the behaviors in detail. It is because our system did not be connected to networks in this our evaluation.

V. Future Works

The current implementation of Alkanet can trace malicious threads injected into legitimate processes. However, there are a multitude of ways beyond thread injection to infect other running processes. For example, there are DLL injection and using hook APIs provided by Windows, such as SetWindowsHookEx. Analyzing such behaviors requires more system calls than those invoked by the malicious behaviors observed by Alkanet. Tracing such behavior may require the analysis of logs to extract system call sequences.

In addition, the current implementation of Alkanet cannot satisfactorily analyze behaviors using network communications. In our tests, we must prevent the active malware being analyzed from performing real attacks against computers and servers actually in use. However, we would rather not filter all packets. The reason is that the purpose of network communications by malware is not only attacks. For example, malware downloads new versions of itself or other malware. One idea for a solution to this problem is to control the malware or its packets to communicate with fake servers or honeypots prepared by us, without interaction with real servers.

There are methods for anomaly detection and malware clustering using tracing logs of system calls. For example, there is a malware clustering method that uses reports output by the automated malware analysis service Anubis [17]. We will evaluate the possibility that tracing logs of Alkanet are practical for use with the existing methods.

VI. Related Works

Traditional dynamic analysis tools run in the same environment as malware. In addition, the tools use services

```
No. [5212, 5213]: Polipos.exe (Cid: 54c.18c) -> svchost.exe (Cid: 480.2c4) (Code Injection)
   No. [5288, 5289]: svchost.exe (Cid: 480.2c4) -> svchost.exe (Cid: 480.22c)
     No. [5959, 5960]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.38c)
     No. [6392, 6393]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.360)
     No. [11340, 11341]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.720)
       No. [14368, 14369]: svchost.exe (Cid: 480.720) -> rundll32.exe (Cid: 220.7f8) (Code Injection)
         No. [14546, 14547]: rundll32.exe (Cid: 220.7f8) -> rundll32.exe (Cid: 220.488)
         ...
     No. [11844, 11845]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.24c)
       No. [15080, 15081]: svchost.exe (Cid: 480.24c) -> alg.exe (Cid: 34c.1c8) (Code Injection)
         No. [15240, 15241]: alg.exe (Cid: 34c.1c8) -> alg.exe (Cid: 34c.5ac)
         ...
     No. [13214, 13215]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.7e0)
       No. [16586, 16587]: svchost.exe (Cid: 480.7e0) -> explorer.exe (Cid: 538.510) (Code Injection)
         No. [16744, 16745]: explorer.exe (Cid: 538.510) -> explorer.exe (Cid: 538.6ac)
         ...
     No. [13802, 13803]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.308)
     No. [14422, 14423]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.2d0)
     No. [14424, 14425]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.a0)
     No. [15144, 15145]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.41c)
     ...
```

Fig. 14. Polipos: thread tree generated by tracing a injected thread and derived threads from it

```
svchost.exe (Cid: 480.41c)
  ...
  [NOT FOUND] No. [15246, 15247]: NtOpenFile \??\c:\program files\netmeeting\drwebase.vdb
  [NOT FOUND] No. [15248, 15249]: NtOpenFile \??\c:\program files\netmeeting\avg.avi
  [NOT FOUND] No. [15250, 15251]: NtOpenFile \??\c:\program files\netmeeting\vs.vsn
  [NOT FOUND] No. [15252, 15253]: NtOpenFile \??\c:\program files\netmeeting\anti-vir.dat
  ...
  [NOT FOUND] No. [15270, 15271]: NtOpenFile \??\c:\program files\netmeeting\avgqt.dat
  [NOT FOUND] No. [15272, 15273]: NtOpenFile \??\c:\program files\netmeeting\lguard.vps
```

Fig. 15. Polipos: scanning files used by anti-virus software

provided by Windows to assist in debugging. However, these tools and services do not run stealthily from the processes being debugged because their purpose is to debug legitimate software. Therefore, malware can detect them easily. To analyze malware that applies anti-debugging techniques, these techniques need to be disabled one by one. Alternatively, analysts may conceal dynamic analysis tools and all detectable vestiges of them. Concealment would include hiding the debugging flags of debuggee, filtering results of the process enumeration API, controlling malware itself and so on. Olly Advanced [18] and Phant0m [19] are plug-ins for OllyDbg debugger [20]. These plug-ins disable many anti-debugging techniques by using ability of debugger or kernel-mode driver. However, it is very difficult to evade all anti-debugging techniques as long as the tools run in the same environment as malware and use debugging assistant services. It is also hard for a general debugger to analyze malware that infects other running processes because general debuggers are only attached to one process.

Therefore, analysts should use other analysis tools and not depend on general debugging assistant services. VAMPiRE [21] implements stealth breakpoints not to be detected by malware. VAMPiRE brings about page faults on arbitrary points purposely and catches and handles the faults. The method of VAMPiRE makes it possible to use breakpoints without using debugging assistant services. VAMPiRE can also execute malware on single step mode using trap flag in EFLAGS register.

Side effects to whole environments executing malware are increased by falsifications to evade anti-debugging techniques. These processes increase overhead and may lead to inaccurate analysis. Analysts and researchers need to use dynamic analysis to reduce side effects to the environment executing malware. Observing from outside of the environment is one effective method. Hence, researchers have implemented dynamic analysis systems based on VMMs or emulators. VMMs and emulators run under a higher privilege level than the operating system in a virtual machine. Such systems can observe all processes and the operating system in virtual machine transparently.

TTAnalyze [22] is automated dynamic analysis system based on QEMU [23]. TTAnalyze executes Windows on an emulated virtual machine and traces system calls and Windows APIs invoked by malware. TTAnalyze outputs report of summarized malware behaviors. Anubis [24], [25] is the successor project of TTAnalyze. However, malware can detect QEMU easily because QEMU emulates specific hardware. Additionally, it is a common approach to implement malware analysis systems based on QEMU. Malware cannot detect Alkanet by similar methods because Alkanet does not emulate specific hardware. In addition, TTAnalyze distinguishes malware by means of process level using values from the CR3 register. Therefore, TTAnalyze cannot trace threads injected to legitimate processes by malware. Alkanet can trace the malicious threads because Alkanet distinguishes malware by means of thread level. Furthermore, the QEMU emulation is inefficient because of the overhead required to emulate the whole environment in software only. Alkanet runs faster because Alkanet is assisted by hardware.

Virt-ICE [26] is a stealth debugger that extends and fixes QEMU for malware analysis. This debugger can analyze not only user-mode malware but also kernel-mode malware. It implements hook functions on the instruction level and the memory access level. Virt-ICE can evade most of anti-debugging techniques. Virt-ICE is effective interactive debugger. Ether [27] is a framework for malware analysis based on Xen [28] virtual machine monitor. The framework can analyze malware transparently using VMM. Ether provides hook functions on instruction level, memory access level and system call level. The vestige of Ether is concealed by itself to not detect by malware. Analysts can implement instruction level trace, system call trace, or arbitrary malware analysis components using the hook functions. In other words, analysts need to implement malware analysis components using the framework to analyze malware actually. In evaluation of Ether, EtherUnpack and EtherTrace had been implemented. EtherUnpack extracts original codes from packed malware. EtherTrace traces system calls invoked by malware. The implementations have been confirmed that they can extract original codes actually, and analyze malware with QEMU detection techniques. In addition, they have been evaluated their performance. However, it has not been shown whether the implementations are possible observing malware behaviors actually. The goal of Alkanet is to analyze malware and summarize malware behaviors automatically.

VII. Conclusion

In this paper, we describe "Alkanet", a malware analyzer using a virtual machine monitor based on BitVisor. Our system can analyze malware within a short time even when the malware applies anti-debugging techniques to evade analysis by dynamic analysis tools. In addition, the analysis overhead is reduced. Alkanet executes malware on Windows XP, and traces system calls invoked by threads. The system call trace logs are obtained in real time via IEEE 1394. Therefore, the system can successfully analyze malware that infects other running processes. In addition, other programs can readily examine the log and process the analysis results to understand the intentions of malware behavior. We confirmed that Alkanet correctly analyzes malware behaviors (e.g. copying itself, deleting itself, creating new processes). We also confirmed that Alkanet precisely traces threads injected to other processes by malware.

In future work, we will make it possible to analyze a multitude of other methods to infect running other processes besides thread injection. We will also implement functions to make detailed observations of malware behaviors using networks. In addition, we will evaluate the possibility that the trace logs of Alkanet are practical for existing anomaly detection or malware clustering methods using system call trace logs.

References

[1] P. Wood, G. Egan, K. Haley, T.-K. Tran, O. Cox, H. Lau, C. Wueest, D. McKinney, T. Millington, B. Nahorney, J. Mulcahy, J. Harrison, T. Parsons, A. Watson, M. Nisbet, N. Johnston, B. Krishnappa, P. Wood, G. Egan, K. Haley, T.-K. Tran, O. Cox, H. Lau, C. Wueest, D. McKinney, T. Millington, B. Nahorney, J. Mulcahy, J. Harrison, T. Parsons, A. Watson, M. Nisbet, N. Johnston, B. Krishnappa, I. Asrar, S. Hittel, E. Chien, E. Park, M. Maniyara, O. Thonnard, P.-A. Vervier, M. Lee, D. Lewis, and S. Wallace, "Internet security threat report volume 17," Symantec Corporation, Tech. Rep., 2012.

[2] N. Falliere, "Windows anti-debug reference," http://www.symantec.com/connect/articles/windows-anti-debug-reference, 2007, Last accessed, July 2012.

[3] M. V. Yason, "The art of unpacking," in *Black Hat USA 2007*, 2007.

[4] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. Kato, "BitVisor: a thin hypervisor for enforcing i/o device security," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. Washington, DC, USA: ACM, 2009, pp. 121–130.

[5] Microsoft, "Standalone and Remote Debugging Tools, Symbols, and Windows SDK," http://msdn.microsoft.com/en-us/windows/hardware/hh852360.aspx, Last accessed, June 2012.

[6] ——, "NtCreateFile function (Windows)," http://msdn.microsoft.com/en-us/library/bb432380.aspx, Last accessed, June 2012.

[7] ——, "SAL Annotations," http://msdn.microsoft.com/en-us/library/ms235402(v=vs.80).aspx, Last accessed, June 2012.

[8] M. Hatada, Y. Nakatsuru, and M. Akiyama, "Datasets for anti-malware research ∼MWS 2011 Datasets∼," in *Computer Security Symposium 2011 (CSS2011)*, 2011, Japanese.

[9] McAfee, Inc., "W32/Sdbot.worm," http://vil.nai.com/vil/content/v_100454.htm, Last accessed, June 2012, 2009.

[10] Symantec Corporation, "Backdoor.Sdbot Technical Details — Symantec," http://www.symantec.com/en/us/security_response/writeup.jsp?docid=2002-051312-3628-99&tabid=2, Last accessed, June 2012.

[11] Trend Micro Incorporated., "PALEVO Worm Leads to Info Theft, DDoS attacks — Trend Micro Threat Encyclopedia," http://about-threats.trendmicro.com/RelatedThreats.aspx?name=PALEVO+Worm+Leads+to+Info+Theft%2C+DDoS+attacks, Last accessed, June 2012.

[12] McAfee, Inc., "W32/Palevo!4D58C671EE49 - Malware - McAfee Labs Threat Center," http://www.mcafee.com/threat-intelligence/malware/default.aspx?id=561341, Last accessed, June 2012.

[13] Sophos Ltd., "49 - 2010 - Threat Spotlight Archive - Threat Spotlight - Security News & Trends - Sophos," http://www-origin-2.sophos.com/en-us/security-news-trends/threat-spotlight/threat-spotlight-archive/2010/49.aspx#f0e736f5-9b72-45c4-a6ec-4cd827fce17a, Last accessed, June 2012.

[14] McAfee, Inc., "W32/Palevo.gen.b!737FE99CE9DB - Malware - McAfee Labs Threat Center," http://www.mcafee.com/threat-intelligence/malware/default.aspx?id=995696, Last accessed, June 2012.

[15] Microsoft Corporation., "Encyclopedia entry: Virus:Win32/Polip.A - Learn more about malware - Microsoft Malware Protection Center," http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Virus%3AWin32%2FPolip.A, Last accessed, June 2012.

[16] Symantec Corporation, "W32.Polip Technical Details — Symantec," http://www.symantec.com/security_response/writeup.jsp?docid=2006-042309-1842-99&tabid=2, Last accessed, June 2012.

[17] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Network and Distributed System Security Symposium*, 2009.

[18] "Olly advanced," http://www.openrce.org/downloads/details/241/Olly_Advanced, 2007.

[19] "PhantOm - Collaborative RCE Tool Library," http://www.woodmann.com/collaborative/tools/index.php/PhantOm, 2009.

[20] "Ollydbg v1.10," http://www.ollydbg.de/, 2010.

[21] A. Vasudevan and R. Yerraballi, "Stealth breakpoints," in *Computer Security Applications Conference, 21st Annual*, 2005, pp. 10–392.

[22] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: a tool for analyzing malware," in *5th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, April 2006.

[23] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX Annual Technical Conference*. Anaheim, CA: USENIX Association, 2005, pp. 41–41.

[24] T. Mandl, U. Bayer, and F. Nentwich, "Anubis - analyzing unknown binaries the automatic way," in *Virus Bulletin Conference*, Geneva, Switzerland, 2009.

[25] "Anubis: analyzing unknown binaries," http://anubis.iseclab.org/, 2010.

[26] Q. N. Anh and K. Suzaki, "Virt-ice: next generation debugger for malware analysis," in *Black Hat USA 2010*, 2010.

[27] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*. Alexandria, Virginia, USA: ACM, 2008, pp. 51–62.

[28] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. Bolton Landing, NY, USA: ACM, 2003, pp. 164–177.