Noname manuscript No.
(will be inserted by the editor)

The Other Guys: Automated Analysis of Marginalized Malware

Marcus Felipe Botacin · Paulo Lício de Geus · André Ricardo Abed Grégio

Received: date / Accepted: date

Abstract In order to thwart dynamic analysis and bypass protection mechanisms, malware have been using several file formats and evasive techniques. While publicly available dynamic malware analysis systems are one of the main sources of information for researchers, security analysts and incident response professionals, they are unable to cope with all types of threats. Therefore, it is difficult to gather information from public systems about CPL, .NET/Mono, 64-bits, reboot-dependent, or malware targeting systems newer than Windows XP, which result in a lack of understanding about how current malware behave during infections on modern operating systems. In this paper, we discuss the challenges and issues faced during the development of this type of analysis system, mainly due to security features available in NT 6.x kernel versions of Windows OS. We also introduce a dynamic analysis system that addresses the aforementioned types of malware as well as present results obtained from their analyses.

Keywords Malware analysis systems · Evasive malware · OS security · 64-bit systems · CPL malware · .Net malware

1 Introduction

Desktop malicious programs continue to threaten users on a daily basis. Despite all security mechanisms and defensive tools, users are still being infected from malicious attachments from phishing e-mail messages, campaigns of malware variants are continuously arising, and operating systems protection, as well as third-party security mechanisms may be subverted by simple, older techniques.

An effective way to understand and work on mitigation of malware infections is to monitor the actions exhibited by suspicious samples in a dynamic analysis system. There are several systems proposed in the literature and available as online services to cope with this task. The latter are a useful resource for researchers, incident response professionals and security analysts, which take advantage of the information presented in the reports to assess specific threats. However, most of the publicly available dynamic analysis systems are only able to run PE32 executable files on Windows XP virtual/emulated machines[1], whereas modern MS-Windows can run PE+ files (64-bits) and .NET framework-dependent samples natively. Another concern of these systems is the ability to analyze samples that require rebooting the OS to complete the infection and, therefore, exhibit important actions of their malicious behavior.

The analysis of this type of malware relies on a system that has to overcome some design and implementation issues related to the security mechanisms present on newer versions of MS-Windows systems, which difficult kernel-based monitoring of subsystems (Registry, file system, processes) in a way more privileged than the malware process. In this paper, we propose a dynamic analysis system to monitor "marginalized" malware, i.e., those samples that have not been addressed by publicly available systems. Our proposed system is able to

Marcus Botacin · Paulo Lício de Geus
University of Campinas, Campinas-SP, Brazil
E-mail: marcus@lasca.ic.unicamp.br
E-mail: paulo@lasca.ic.unicamp.br

André Ricardo Abed Grégio
Federal University of Paraná, Curitiba-PR, Brazil
E-mail: gregio@inf.ufpr.br

[1] As detailed on Section 4.4

analyze PE32 (Portable Executable 32-bits), PE+ (PE 64-bit version), CPL (Control Panel) and .NET/Mono files in Windows 7 and 8 environments, as well as to monitor reboot-activated malware and extract their behavior. The main contributions of this paper are: (i) we provide an overview on CPL and .NET/Mono file types and discuss their role in current malware attacks; (ii) we present the challenges of designing and implementing a malware dynamic analysis system using modern Windows operating systems (7 and 8 — kernel NT 6.x); (iii) we introduce a new dynamic analysis system to monitor both 32 and 64-bits malware samples, which is able to analyze reboot-activated malware, as well as samples encapsulated in other executable file formats (specifically CPL and .NET/Mono). Our work provides richer behavioral information than other available systems, showing how current malware samples act and helping in a better understanding of them.

The remainder of this paper is organized as follows: In Section 2, we provide a background on different file types being used to deploy malware (PE+, CPL, and .NET/Mono files), on new Windows internal mechanisms and their implications on current in-system monitoring tools, and we present the related works; in Section 3, we detail the architecture of our proposed dynamic analysis system and choices about its design; in Section 4, we present some case studies to evaluate our proposed system using malware samples collected in the wild, mainly the ones marginalized by currently available public analysis systems; in Section 5, we discuss the limitations of our system, and, finally, in Section 6, we provide our final considerations.

2 Background and Related Work

We present a brief background about file types used by malicious programs, simple techniques malware uses to prevent analysis, dynamic analysis systems, and tools from literature and the monitoring techniques they make use of.

2.1 File Encapsulation

Executable file formats that run correctly in a victim's system may hinder the dynamic analysis, if used to encapsulate malware. We have observed a significant number of CPL and .NET binaries in our dataset (we collected these samples in the wild during the last few years), apart from the prevalence of PE files. Below, we discuss the file types available among our samples.

CPL. It is the file extension of Control Panel applets, which are dynamic libraries (DLL) that export the *CPlApplet* function [28] and can be directly executed by users' double-click, often leading to a background execution. CPL malware is a Brazilian phenomenon mostly related to Internet Banking attacks, whose actions are recognized by antivirus vendors as significant threats [21].

.NET/Mono. Mono is a software platform that allows the creation of cross-platform applications (`http://www.mono-project.com/`) and it is an open source implementation of Microsoft's .NET framework. This kind of sample has the same range of actions that traditional PE ones, but it usually crashes on or does not work on .NET framework versions for Windows XP (used in most publicly available sandboxes). We observed an increase in the number of malware samples using this file type[2], but did not find specific literature about them.

PE+. Portable Executable [35] is a file format used on Windows systems mainly for executable and library files. There are two versions of it: PE32 and PE+, for 32 and 64-bits files, respectively. Analysis support for PE32 is widespread, but there is a lack of specific PE+ malware analysis tools. As reported in [5], 64-bits malware is an *inevitable move*. The more 64-bits platforms are deployed, the more it becomes attractive for malware creators. Malware developers do not need to move immediately from coding 32 to 64-bit malicious programs—modern Windows are still compatible with both (WoW64 [26] is responsible for such translation)—, but if they do, researchers must be able to analyze those samples, since 64-bits malware is already in the wild: in [5], the authors present a version of the Zeus malware that is able to inject 32 and 64-bits versions of itself, depending on the host; Regin [18] samples also have 64-bits modules. In spite of that, there is still a lack of malware analysis literature for 64-bits samples' analysis and execution. For instance, a reference textbook [40] covers PE Header changes and the new 64-bit calling convention but do not address changes in dynamic execution. Furthermore, as far as we know, there are no portable and publicly available sandboxes able to adequately handle PE+ files in the context of malware analysis[3]. Thomas et al. [43] discuss some differences of executing 32-bits samples on 64-bits modern operating systems (from Windows 7 forensics procedures perspective), while Corregedor et al. [6] provides a first step on analyzing a 32-bit modern Windows, which makes us to consider the results on 64-bits ones.

[2] We identified as .NET 0.6%, 1.1%, and 7.6% of all samples from our dataset collected in 2013, 2014, and the first quarter of 2015, respectively.

[3] Solutions like Sandboxie (`http://www.sandboxie.com`) are not designed for this purpose and can be detected due to their userland modules.

2.2 Dynamic Analysis Evasion and Reboot

Malware creators use a varied set of techniques to bypass sandbox-based analysis, such as virtual machine detection and fingerprinting [16,34,12]. Different approaches are required to address them, such as the monitoring of split behavior [2] and the use of bare metal systems [15]. An effective evasive technique against dynamic analysis system is to delay the execution of a sample, either by stalling it or requiring a reboot to complete the process of performing malicious actions. Both approaches have high chances of success against ordinary users and bypass automated sandboxes and bare metal systems. The stalling malware problem is addressed by [20].

Lindorfer et al. proposed a method to trace the evolution of malware samples over time [19], which consists on running self-updating malware every day and analyzing the downloaded artifacts. Their approach uses a snapshot patch to restore sandbox infection from the previous day, since their goal is to observe behavioral changes in already deployed malware. On the other hand, our approach intends to observe the behavior of malware that requires a reboot as a way to evade dynamic analysis.

2.3 Dynamic analysis techniques and systems

In this section, we cover techniques used to monitor malware execution, based on [9].

SSDT Hooking. This technique allows the interception of system calls by directly changing the System Service Dispatch Table (SSDT) pointer to a trampoline function. This approach runs in a more privileged ring (kernel land) than the monitored sample (userland).and is able to monitor the system in a wide manner, without the need to handle each process individually. Kirat et al. [15] present Barebox, a dynamic analysis system that does not use a virtual or emulated environment. It monitors the actions of the analyzed samples by hooking SSDT while executing them in a bare-metal environment. It allows the execution and observation of the behavior of malware that employs anti-VM techniques. Afonso et al. [1] present a framework able to analyze drive-by downloads. If an URL causes the download of an executable, the framework's OS module runs and monitor its actions. This module consists of a pool of analysis machines (emulated and bare-metal) that make use of an SSDT hooking kernel driver.

DLL Injection. In this case, hooks for system calls are implemented through the injection of a Dynamic Linked Library (DLL) inside the process to be monitored. The monitoring DLL loaded inserts a JMP instruction to the trampoline function in order to redirect the execution flow. Some drawbacks include it being userland-based (more prone to be bypassed) and requiring the injection of the DLL in each of the to be monitored processes. Examples of tools employing this techniques are CWsandbox [46] and Cuckoo [11].

Virtual Machine Introspection. Consists of instrumenting the virtual machine hypervisor to collect information about the guest system. It allows for finer-grained analysis, since it is possible to obtain assembly or IR instructions. It is also more secure (from the monitoring perspective), since it is external to the malware execution environment. The main disadvantage of this technique is the dependence on the guest OS, which makes it difficult to bridge the semantic gap [33][8] and does not allow using bare metal systems. Anubis [3] is an analysis system that implements VMI (Virtual Machine Introspection) through Qemu instrumentation [4]. This technique creates a layer between the analysis system (guest) and the processing environment (host). Its use enables monitoring actions performed within the analysis environment without any interference in the environment where the malware is running or in the guest system. As far as we know, Anubis analysis environment is based on a Windows XP operating system. Modern VMI approaches, such as hardware-assisted hypervisor monitoring, could also be used to inspect Windows x64 kernel without restriction, since they operate from outside of the system. Nevertheless, general approaches like Ether [7] and HyperDbg [10] are based on Windows XP, depending of a port for Windows 7 and 8 in order to handle both OS semantic gap. Another approach, CXPInspector [47], was developed for Windows 7 x64 and is able to handle the OS kernel semantic gap. There are commercial solutions that aim at analyzing 64-bit malware, but they are mostly based on this kind of external instrumentation. For instance, Vmray relies on VMI [4], while Lastline's analyzer depends on full emulation [17].

System Callbacks and File system Filters. In this technique, the monitoring of a system uses OS-defined callbacks and file system intercepts. Every time a given action is performed on the system, a registered-callback is called. This approach also runs on ring 0 (kernel level) and is able to monitor the whole system without injecting process individually. On the one hand, it does not tamper with any kernel structure, but on the other hand, it is limited to OS-provided subsystem monitoring. An example of tool that employs callbacks is CaptureBAT [39]. Capture-BAT is a kernel driver based malware analysis tool, which runs on Windows XP SP2 and monitors the file operations of "

[4] https://www.vmray.com/technology/

Read" and "Write" using a file system filter, operations of "SetValueKey" and "DeleteValueKey" on Registry using kernel callbacks, and the creation and termination of processes using the same technique. However, unlike previously mentioned tools, Capture-BAT has no processes tracking, so all information capturing is performed in system-wide mode. Furthermore, the collected information is very restricted (timestamp, source process, target key, file or process).

Mixed Approaches There are solutions that combine multiple approaches to overcome system's limitations. Sandboxie, for example, makes use of a kernel driver along with userland DLLs[5].

2.4 Changes in Security Mechanisms

Regarding defensive mechanisms, Microsoft Windows evolved from XP (kernels NT 5.1 and 5.2) to Vista, 7 and 8 (kernel NT 6.x). Windows 8 is claimed to be the most secure version of this operating system (apart from the recent release of Windows 10) due to its new security features. We briefly describe how they impact the implementation of malware dynamic analysis systems.

Kernel Patch Protection. This mechanism prevents kernel-mode drivers from extending or replacing kernel services in undocumented ways, aiming to increase the security of the OS against rootkits. KPP is only present in 64-bits versions of Windows 8. Enabling KPP has an immediate consequence for dynamic analysis, since it prevents the use of hooking techniques (e.g., SSDT).

Driver Signing. Its goal is to prevent arbitrary components from loading into kernel space. This approach may be restrictive for analysis systems based on monitoring drivers, due to the price of a certificate and the signing process conditioned to Microsoft Quality Process approval. However, this protection can be turned off for analysis purpose, allowing us to load a malware analysis driver. This workaround was already deployed by other tools, such as Sandboxie[6].

Session Isolation. This mechanism aims to isolate different families of applications, such as graphical applications, user jobs, system services and remote jobs to keep data and application privileges restricted to their own (user) space. This restriction prevents processes from writing on foreign regions (at least in different sessions) without user's direct control, as seen on `CreateRemoteThread` [30]. Analysis tools based on DLL injection may be bypassed by a sample whose payload is installed as a service running on a different session. For instance, Cuckoo sandbox replaces `CreateRemoteThread` with `QueueUserAPC` [24] to prevent bypass, requiring a trap to inject the monitoring payload. However, this opens another possibility of evasion for the sample.

API changes. Newer Windows versions also introduced new API interfaces. Yet the legacy interfaces are still supported, we have to implement our analysis system using the newer ones to take advantage of performance gains and support new operation modes and tools. As an example of API change, CmRegisterCallback [31] was updated to CmRegisterCallbackEx [32].

WoW64. As mentioned before, 32-bits samples can run on 64-bits systems through an address translation performed by the WoW64 (Windows 32-bit on Windows 64-bit) subsystem [26]. Due to WoW64 translation, we can monitor the whole system without restrictions and/or a special handle for each process type, but 32-bits malware is able to threaten users of 64-bits systems. Hence, it brings the entire already established 32-bits threats' arsenal to 64-bits systems.

3 System Overview

The design of a dynamic analysis system depends on implementation choices. In this section, we present our decisions and architecture.

3.1 Project Decisions

This section presents how the new security mechanisms being used by Windows influenced in our project decisions. One of the main requirements of an analysis tool is to be portable for other versions of the target operating system. While SSDT hooking used to be a well-established technique for finer-grained in-guest control of the system calls and program flow, it is not allowed anymore on 64-bits versions of Windows 7 and 8 due to KPP. DLL injection-based tools are very detectable and of limited application due to the isolation provided by the aforementioned OS versions. The main advantage of VMI is the possibility of out-of-the-guest monitoring, but its deployment on different OS versions depends on hard effort to re-instrument the virtualization layer and port the system call interception solution. The use of callbacks and filters to develop monitors is the recommend development solution, but these techniques rely on API limited by the OS vendors. However, their current capabilities are enough to develop monitors suit-

[5] `http://www.sandboxie.com/index.php?ContributedUtilities#BlockProcessAccess`

[6] `http://www.sandboxie.com/index.php?ExperimentalProtection`

able for dynamic analysis of malware, so we chose them to implement our proposal.

Most of the described OS security mechanism can be subverted, for instance, it is possible to defeat the driver signing requirement [13] or to break the KPP [41, 42, 36]. In fact, there are malware samples that had already deployed some exploitation techniques against these protection mechanisms: Regin [18] uses a signed driver to load itself as a rootkit; TDL [38] does this by using a bootkit to disable driver signature requirement; the Equation Group [14] exploits known legitimate drivers bugs to accomplish privilege escalation. These exploits are version-dependent, violating our requirement of portability. Besides, they can be patched any time by system vendors. Due to this "instability", we opt out from implementing our tool by relying on exploiting the target OS. Instead, we adopted system callbacks and filters. While the use of those were already suggested by [37], we expand our implementation to include more capture capacities in addition to file system filtering.

3.2 Architecture Overview

Our system is based on three main pipeline stages that are responsible for *collection*, *analysis* and *results presentation* of processed malware samples. First of all, we collect the available samples (through phishing, honeypots, Web sites or donations from collaborators) and store it in a database. Then, for each sample not yet analyzed, we create a task that waits for further processing to produce a report. The analysis subsystem has a scheduler, external analyzers, virtual machines and a results repository. The scheduler is the module that interacts with the collection interface to receive unprocessed tasks and dispatch them to all registered analysis machines, providing scalability and concurrency control. If everything is OK, this subsystem launches the external analyzers (to obtain static information and network traffic) and the dynamic analysis environments.

The dynamic analysis environment consists of pristine installations of Windows 7 or 8 with the monitoring driver running inside. It can be bare-metal, virtualized or emulated. For this paper, we set up an emulated environment based on Qemu/KVM. The monitoring driver is composed by three modules—Registry callback, process callback and file system filter. There is also an external client responsible to obtain the internal captured data, filter it as defined by flags passed as arguments (e.g., PID tracking, system-wide monitoring), and produce the resulting log files. Figure 1 shows an overview of the proposed system.

For the sake of flexibility and portability, we chose to deploy our monitoring tool as a driver compiled for Windows 7 and 8 from the same source. This driver consists of a basic toolset and three modules for intercepting malware action. The basic toolset handles the following information:

- **Timestamps**: the system's time and date (with millisecond precision) are retrieved through `KeQuerySystemTime`, which is called every time a callback is activated. Since it is implemented as a non-reentrant function, it provides the order of the monitored actions.
- **Processes**: PID and process name are retrieved on every callback (once the caller is logged) through `ZwQueryInformationProcess`, while process handlers are taken from PID through the call of `PsLookupProcessByProcessId`.
- **Objects**: we use `ObQueryNameString` to retrieve file system paths from objects pointed by handlers passed as arguments in callbacks.
- **Lists**: After captured, data is buffered on a FIFO queue implemented using kernel list facilities. The data is only sent to userland when a `DeviceIoCtl` call is made by the driver client.
- **String manipulation**: to allow the driver to handle communication from distinct clients (and do not depend on them), we format the output directly in kernel. All data, but integers, were converted to a `UNICODE_STRING`. We used the safe library `NTSTRSAFE` to correctly handle strings in kernel.
- **Serialization**: collected data is stored serialized in a kernel list as a bit stream. The most important data structure serialization is that applied to `UNICODE_STRING`, which is a structure containing a pointer to data and its size. The serialization operation copies the data pointed by `UNICODE_STRING` directly to client structure along with its size.
- **I/O handling**: these operations are performed using IOCTLs [22]. The driver is able to handle and export the IOCTL codes for system configuration, such as starting and stopping the monitoring; enabling or disabling debugging mechanisms, as well as for obtaining collected data.

The aforementioned intercepting of actions is performed by the three modules described below:

- **Registry Monitoring**: the Registry callback (registered by `CmRegisterCallbackEx` data consists of two arguments, a data type, which represents possible actions [25], and a pointer cast to the data type;
- **Process Monitoring**: this callback relies on `PsSetCreateProcessNotifyRoutine` with the pro-

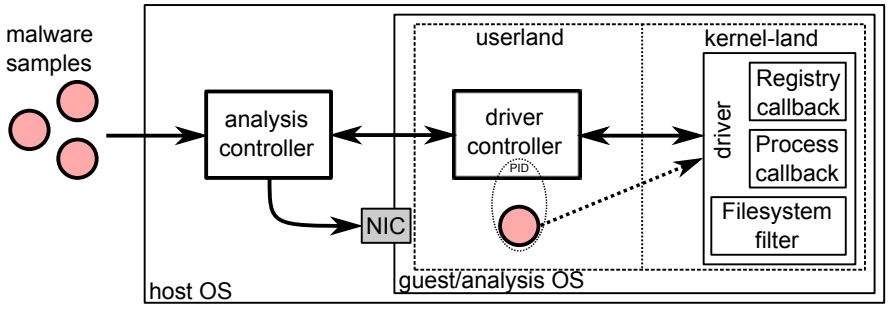

Fig. 1 The analysis controller fetches enqueued malware samples and send the first to an idle analysis machine through the driver controller; it loads the kernel driver, obtain the sample's PID and executes it on userland; meanwhile, the analysis controller captures the produced network traffic. Dense arrows illustrate the monitoring flow; the sample on userland has its actions intercepted by the driver through its PID.

cess handler and a boolean value (type of action, such as *create* or *delete* process) as arguments.

- **File System Monitoring**: This monitor is implemented by a file system filter, installed through the call of `FLTRegisterFilter`, which intercepts every system IRP [23] call. We chose to filter the action before it happens (pre-operation) to monitor even unsuccessful operations. We also collect additional data when a deletion is performed:
  - **Deleted Files Monitoring**: Once an `IRP_MJ_SET_INFORMATION` is identified, we look at its file information class to determine if it is a delete operation. If so, the target file is copied for further offline analysis.

3.3 Tests and Validation

We tested our proposed system to verify its monitoring capabilities against actual malware whose actions were previously known. The listings below show actions that can be monitored and the format used to log information.

Listing 1 illustrates a sample writing its own path on the `\Run` Registry key, which allows it to survive reboots (a typical persistence behavior). Listing 2 shows a sample deleting a Registry key related to the Internet Explorer settings.

Listing 1 Registry's write operation log example.

```
<timestamp>|SetValueKey|2032|C:\7G6C5n.exe|\
    REGISTRY\USER\S-...-1001\Software\Microsoft\
    Windows\CurrentVersion\Run|SoftBrue|"C:\7G6C5n.
    exe"
```

Listing 2 Registry's delete key operation log example.

```
<timestamp>|DeleteValueKey|3028|C:\visualizar.exe|\
    REGISTRY\MACHINE\SOFTWARE\Wow6432Node
    \Microsoft\Windows\CurrentVersion\Internet Settings
    \ZoneMap|IntranetName|
```

Listing 3 illustrates a sample (`visualizar.exe`) creating a process (`dll.exe`) and the WoW64 mechanism acting to translate this request. Listing 4 shows a process (`rundll32.exe`) terminating another one (`wimplayer.exe`). This type of operation can be used to terminate an update or defense mechanism, such as a known antivirus.

Listing 3 Process' create operation log example.

```
<timestamp>|CreateProcess|3028|C:\Monitor\Malware\
    visualizar.exe|2440|C:\Windows\SysWOW64\dll.exe
```

Listing 4 Process' terminate operation log example.

```
<timestamp>|DeleteProcess|2548|C:\Windows\SysWOW64
    \rundll32.exe|3040|C:\ProgramData\blackberry\
    wimplayer.exe
```

Listing 5 illustrates the process `visualizar.exe` writing a file on disk. It may be the result of a download, a drop, keylogger data, Trojanizing an existing file, virus-related data appending, ransomware activity etc. Listing 6 illustrates the malware sample `deposito.exe` deleting a file, a common action performed for anti-forensics.

Listing 5 Filesystem's write operation log example.

```
<timestamp>|WriteOperation|3028|C:\visualizar.exe|C:\
    Windows\SysWOW64\dll.exe|
```

Listing 6 Filesystem's delete operation log example.

```
<timestamp>|DeleteOperation|2032|C:\deposito.exe|C:\
    ProgramData\rr.txt|
```

Listing 7 illustrates the network behavior of a sample fetching a file through an HTTP GET request, while Listing 8 illustrates a sample sending victim's information through an HTTP POST request. These request types may be used for drive-by downloading and/or information stealing, for example.

Listing 7 Network traffic monitoring log example (GET).

```
<timestamp> <Analyzer IP> XX.YY.ZZ.121 HTTP 290
    GET /.swim01/control.php?ia&mi=00B5AB4E-47098
    BC3 HTTP/1.1
```

Listing 8 Network traffic monitoring log example (POST).

```
<timestamp> <Analyzer IP> XX.YY.ZZ.43 HTTP 335
    POST /notas/nota6/index.php HTTP/1.1 arq
    =07/04/2014 / 13:03:52 / &condicao=WIN8_VM1 [
    Brasil]
```

3.4 Overhead Measurement

To minimize the overhead of intercepting and logging samples' actions, we opted for a decoupled data acquisition procedure (illustrated on Figure 2). We measured the impact penalties imposed by our monitoring driver to the system during automated analysis. To do so, we coded programs that perform specific actions on each monitored subsystem and ran them 100 times in order to get a decimal significance in the standard deviation. These programs simply write and read the Registry or the file system, and create and terminate a suspended process[7]. The tests were performed on a KVM virtualized machine without any other load than the testing process. Table 1 shows the average of the measured results.

4 Case Study

We used our system to analyze the behavior of samples not handled by other available malware analysis systems: .NET/Mono, CPL, and PE+ files. For each type of malware, we observed the following actions:

- **Run keys (persistence)** - the monitored sample wrote itself in an "AutoRun" related Registry key;
- **BHO injection** - a write in the Browser Helper Object key to install a plugin.
- **Proxy settings** - the sample re-route the browser gateway by adding a Proxy Auto Config (PAC) file in the Registry.
- **IE keys** - the browser changed any configuration registry key related to Internet Explorer (default browser).
- **Firewall settings** - modification in this security mechanism key.
- **Enable file trace** - the sample uses a log file to determine its correct execution, indicating possible evasive mechanisms.
- **Creation** - the sample launched a new process.
- **cmd.exe launching** - creation of the system's shell.
- **Termination** - the sample stopped a running process.
- **Services** - the sample launched `services.exe` utility to install itself as a service, load, or unload a kernel driver.
- **Delete** - the sample removed a file from the target system.
- **Write a PE** - the sample wrote in a portable executable file (existing or new).
- **Internet settings** - the sample changed disk files related to Internet Explorer settings, such as history and cache.
- **Write .sys** - the sample wrote a driver file on the target system's disk.

Dataset: Our samples were collected in the wild in the last years from malicious phishing attachments, infected users, and honeypots. This collection allowed us to identify the growth of CPL and .Net malware, shedding light on these threats. In addition, we enriched out dataset with downloaded PE+ files.

Below, we present the resulting actions obtained from the analysis of the different file types samples;

4.1 .NET/Mono.

We ran 426 samples (from suspicious e-mail attachments collected between 2012 and 2015) in our system and evaluated the obtained analysis logs. Table 2 shows the occurrence of the aforementioned actions.

In general, we observed two major types of "predominant" behavior on .NET samples: information stealing, accomplished by re-routing the user to malicious domains through the modification of Internet Explorer settings, installation of Browser Helper Objects extensions, or a proxy in the Registry, changes in cookies, bookmarks and history; downloading of additional components, such as executable or driver files.

4.2 CPL.

We randomly selected 1,700 CPL samples from our database to observe their actions. In general, their log footprints were small (Table 3) and their names contained terms from Internet Banks. We have checked around 10% of them manually and confirmed that they were malicious bankers, thus requiring manual stimulation input from the victim or the download of another component. After that, we sent all samples to VirusTotal and verified that the two most returned AV labels

[7] We measure suspended processes to avoid penalties from external factors.

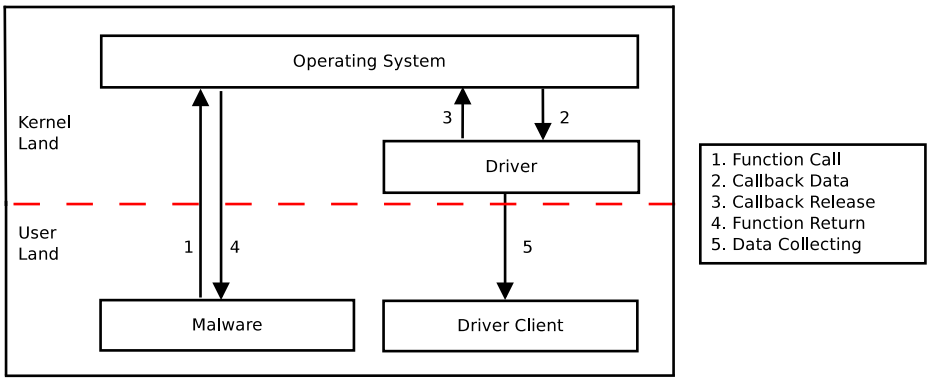

Fig. 2 Malware monitoring data acquisition process— interception and logging.

Table 1 Average overhead imposed by monitoring a sample in a controlled experiment.

Operation type	Without the driver	With the driver	Avg. overhead
Registry	5.13×10^{-2} ms	5.47×10^{-2} ms	6.65%
Process	1.02 ms	1.14 ms	11.76%
File	4.06×10^{-1} ms	5.86×10^{-1} ms	44.27%

Table 2 Suspicious behavior observed on 220 .NET/Mono analyzed samples.

Registry actions			
Run keys (persistence)	125 (29.34%)	BHO injection	3 (0.7%)
Proxy settings	127 (29.81%)	IE keys	139 (32.63%)
Firewall settings	2 (0.47%)	Enable file trace	269 (63.15%)
Process actions			
Creation	274 (64.32%)	cmd.exe launching	19 (4.46%)
Termination	96 (22.54%)	services	18 (4.24%)
File system actions			
Delete	145 (34.04%)	Write a PE	223 (52.35%)
Internet settings	197 (46.24%)	Write .sys	47 (11.03%)

were Downloader (1,245) and Banker (1,148). An inspection of the captured network traffic revealed that 1,110 samples performed at least one HTTP request related to downloading objects or sending system/user data.

Most of the collected CPL files behaved similarly to the .NET ones. However, they ran in a more silent way, i.e., in background and without showing any dialog box or opening the browser. The amount of terminated processes occurs because as CPL files are injected, they finish some instances of their own `rundll32` process. In addition, we observed that most registered paths under Registry "Run" keys refer to downloaded payloads.

4.3 PE+

We collected the 998 64-bits samples from VirusShare (`http://virusshare.com/`), from which 970 (97,2%) were successfully analyzed, i.e., did not crash the environment and returned non-empty analysis logs. Below, we compare our analysis results with publicly available reports about known 64-bits malware.

4.3.1 Wootbot.

According to Wootbot report [29], this threat:

1. creates a copy of itself in %windir% or the system folder;
2. may add a value to one or more Registry keys, which causes the worm to run each time Windows starts. Some variants create a Windows system service for this purpose;
3. can connect to an internet relay chat (IRC) server;
4. can exploit a Windows Local Security Authentication Server (LSASS) service process vulnerability to create a command shell on a remote computer.

Our proposed system was able to log operations related to items 1, 3, and 4 above (Listing 9, Listing 10, and Listing 11, respectively).

Listing 9 Writing itself on Windows Location.

```
<timestamp>|WriteOperation|1612|C:\Users\Win7\
    AppData\Local\Temp\IXP000.TMP\server.exe|C:\
    Windows\SysWOW64\server.exe|
```

Table 3 Monitored operations observed from CPL malware samples.

Registry actions			
Run keys (persistence)	61 (3.59%)	BHO injection	3 (0.18%)
Proxy settings	114 (6.71%)	IE keys	131 (7.71%)
Firewall settings	7 (0.41%)	Enable file trace	56 (3.29%)
Process actions			
Creation	111 (6.53%)	cmd.exe launching	12 (0.71%)
Termination	879 (51.71%)	services	1 (0.06%)
File system actions			
Delete	73 (4.29%)	Write a PE	860 (50.59%)
Internet settings	587 (34.57%)	Write .sys	5 (0.29%)

Listing 10 Opening the LSASS process.

```
<timestamp>|CreateOperation|828|C:\Windows\System32
    \svchost.exe|C:\Windows\System32\lsass.exe|
```

Listing 11 IRC traffic captured.

```
<timestamp> IP <Analyzer IP>.49184 > xx.yyy.zzz.76.
    ircd: Flags [S], seq 1351309149, win 8192, options [mss
    1460,nop,wscale 8,nop,nop,sackOK], length 0
```

4.3.2 Jorik.

According to Jorik report [27], this malware:

- creates %windir%\assembly\ nativeimages_v4.0.30319_32\temp\998-0\ system.data.datasetextensions.dll on affected computer;
- creates the file `update.vbe` on the victim.affected computer;
- may contact a remote host.

In our logs, we verified that the malware uses the `svchost.exe` process to check if a variant is present on the victim's system (see Listing 12 and Listing 13). Listing 14 shows the malware trying to resolve the domain `downloads.fcuked.me.uk` through Google DNS (probably to download a missing component), but it was not found at that moment.

Listing 12 Another Infected Process

```
<timestamp>|CreateProcess|1456|C:\Monitor\Malware\
    VirusShare_0826f81f22867a464021aa2f94576693|576|C:\
    Users\Win7\AppData\Local\Temp\IXP000.TMP\
    svchost.exe
```

Listing 13 Windows directory writing

```
<timestamp>|ReadOperation|576|C:\Users\Win7\AppData
    \Local\Temp\IXP000.TMP\svchost.exe|C:\Windows\
    assembly\NativeImages_v2.0.50727_64|
```

Listing 14 Network Traffic

```
<timestamp> IP <Analyzer IP>.61489 > google-public-
    dns-a.google.com.domain: 24872+ A? downloads.
    fcuked.me.uk. (40)
```

4.3.3 DarkKomet.

According to DarkKomet report [44], it:

1. drops following copies of itself into the affected system (\%Application Data\ %HostProcess\malware name.exe);
2. creates \%Application Data%\dclogs;
3. adds Registry entries to enable its automatic execution at every system startup;
4. disables Windows firewall.

We were able to observe all of the expected actions: Listings 15 and 16 show the logs representing item 1 above; Listings 17, 18, and 19 illustrate the actions mentioned on items 2, 3, and 4, respectively.

Listing 15 DarkKomet creating/calling other processes.

```
<timestamp>|CreateProcess|2536|C:\Monitor\Malware\
    VirusShare_27b831e93697d45d978796198b421033|2952|
    C:\Users\Win7\AppData\Local\Temp\IXP000.TMP\
    andrew.exe
<timestamp>|CreateProcess|2952|C:\Users\Win7\AppData
    \Local\Temp\IXP000.TMP\andrew.exe|2820|C:\Users
    \Win7\Documents\MSDCSC\msdcsc.exe
```

Listing 16 DarkKomet copying itself locally.

```
<timestamp>|CreateOperation|2952|C:\Users\Win7\
    AppData\Local\Temp\IXP000.TMP\andrew.exe|C:\
    Users\Win7\AppData\Local\Temp\IXP000.TMP\
    andrew.exe.Local|
```

Listing 17 Log file creation.

```
<timestamp>|CreateOperation|2820|C:\Users\Win7\
    Documents\MSDCSC\msdcsc.exe|C:\Users\Win7\
    AppData\Roaming\dclogs|
```

Listing 18 DarkKomet changing Registry Autostart key to persist.

```
<timestamp>|SetValueKey|2952|C:\Users\Win7\AppData\
    Local\Temp\IXP000.TMP\andrew.exe|\REGISTRY\
    USER\S
    -1-5-21-603313242-1591760659-684491057-1000\
    Software\Microsoft\Windows\CurrentVersion\Run|
    MicroUpdate|C:\Users\Win7\Documents\MSDCSC\
    msdcsc.exe
```

Listing 19 DarkKomet disabling Windows Firewall.

```
<timestamp>|SetValueKey|2820|C:\Users\Win7\
    Documents\MSDCSC\msdcsc.exe|\REGISTRY\
    MACHINE\SYSTEM\ControlSet001\services\
    SharedAccess\Parameters\FirewallPolicy\
    StandardProfile|EnableFirewall|0 0 0 0
```

4.3.4 ZBot.

According to Zbot report [45], this malware:

1. connects to `http://checkip.dyndns.org` (observed in Listings 20, 21, and 22);
2. requires the existence of a random executable's file name on the following directory to properly run: C:\Users\user name\AppData\Roaming\ (observed on Listing 23).

Listing 20 Connection to `checkip.dyndns.com`.

```
<timestamp> <Analyzer IP> <checkip.dyndns.com IP>
    HTTP 148 GET / HTTP/1.1
```

Listing 21 Content of file fetched from previous GET.

```
Current IP Address: <Analyzer gateway IP address>
```

Listing 22 Removal of fetched file.

```
<timestamp>|DeleteOperation|1852|Trojan-Spy.Win64.
    Zbot.a|C:\Users\User_Windows_VM\AppData\Local\
    Microsoft\Windows\Temporary Internet Files\Content
    .IE5\SP89WLWJ\dyndns[1].txt|dyndns[1].txt
```

Listing 23 Random file name created on "Roaming" directory.

```
<timestamp>|CreateOperation|1852|Trojan-Spy.Win64.
    Zbot.a|\Users\User_Windows_VM\AppData\Roaming
    \Gevoun\riod.exe|
```

4.4 Systems Comparison

In order to verify whether publicly available malware analysis systems are able to process marginalized malware, we submitted some samples through their Web interfaces. These systems[8] either warned the user that the file type is not supported, or did not return any results at all. Table 4 presents the results of trying to analyze executables encapsulated with other file types, in addition to reboot-dependent samples. The results show that our solution is the only one able to handle PE+ executables and that combines CPL and .Net analysis support to provide more information about these types of infection.

4.5 Surviving Reboots

We analyzed 2,937 malware samples in our system collected from phishing e-mail attachments and reported incidents in 2015. From this total, 399 (13.59%) wrote their own binary path on AutoRun-related Registry keys, consequently triggering our "reboot-aware" analysis feature. In this mode, our driver is installed as a bootkit, hence capturing system-related data in a wider manner since the system's startup.

We were able to observe that 167 (59.86%) of them presented network traffic capture files larger than those obtained in the first run (i.e., before reboot). Moreover, we were able to identify malicious activity after reboot, sometimes performed by the same executable inserted first time.

It is worth to mention that tracing a previously installed sample's activities is not a simple task. Since we do not have a PID after the analysis system's reboot, we had to perform capture in a system-wide manner. This way, the analysis system produces a much larger log file that needs to be parsed in order to remove noise. Hence, monitoring reboot-dependent malware is an interesting way to detect behavior that does not appear during the first run of a sample, as well as persistent malware behavior. As a drawback, it requires additional efforts on results post-processing. Table 5 reveals information about the analysis of the 167 samples, before and after the reboot.

A careful analysis showed us that all samples performed a single-step infection on their first run, i.e., they installed themselves on AutoRun related Registry keys. According to labels obtained from VirusTotal, 73% of the samples target Internet Banking users (they are labeled as "bankers"). This causes them to wait for any Internet Banking related input, such as the user typing one URL from a list of bank sites, in order to redirect the victim to a fake/cloned site through

[8] `http://anubis.iseclab.org`, `https://malwr.com`, `http://www.threatexpert.com`, `http://camas.comodo.com`, `http://www.threattracksecurity.com/resources/sandbox-malware-analysis.aspx`

Table 4 Comparison among publicly available dynamic analysis systems and our proposed system regarding other than PE32 file types.

Sandbox/File type	PE32	PE+	.NET	CPL	Reboot
Anubis	✓	✗	✓	✗	✗
Cuckoo	✓	✗	✗	✓	✗
ThreatExpert	✓	✗	✗	✗	✗
Camas Comodo	✓	✗	✗	✗	✗
CWSandbox	✓	✗	✗	✗	✗
Our system	✓	✓	✓	✓	✓

Table 5 Actions observed during analysis before and after the reboot for reboot-dependent malware (BHO and services actions omitted due to non-occurrence).

Type	Registry					Process			File			
Action	Run	Proxy	Firewall	IE	trace	Create	Term.	cmd	Del.	Net	PE	.sys
Before	100%	46.11%	0.60%	47.31%	1.80%	63.47%	50.30%	3.59%	12.57%	83.23%	95.21%	0.60%
After	4.19%	0%	0%	0%	0%	27.54%	0%	0%	7.78%	46.11%	0.60%	0%

the proxy auto configuration file installed previously, in which we found the name of known banks.

5 Limitations

Our proposal is extend dynamic analysis systems to extract more knowledge about malware samples whose usage is increasing, but for which the automated analysis is left aside. While we showed that such analysis is viable, it may suffer from issues that are innate for this kind of approach. Below, we discuss some limitations of our approach and possibilities to overcome them.

Since the monitoring mechanism is based on callbacks and a file system filter, there is an inherent limitation of using only the APIs provided by system manufacturer. However, there is a trade-off between the ability to intercept events and the lack of permission to modify the kernel of modern Windows operating systems. There are user-level approaches that do not rely on any kernel modification, which can be applied to improve the analysis process and may be used in the future if necessary.

The use of virtualized/emulated environments is a natural choice for scalability. However, some malware samples are able to identify virtualized environments, hiding its malicious behavior. In order to overcome this limitation, we can run samples on bare-metal environments. It is worth noticing we met the portability requirement to ensure it is possible. As a kernel driver, our analysis system could be easily deployed on a bare-metal environment without any modification. Then, we would have to manage the restore procedures and a greater overall time for analysis. In spite of the ability of our kernel-based system to handle userland malware, it may be subverted by rootkits and other privileged threats.

Furthermore, the nature of our dataset—with many banker samples—imposes a limitation related to the required stimulation to trigger malware functions during the execution. Other limitations include evasive malware, i.e., which employs anti-analysis techniques, such as sleeping/stalling features and the need of rebooting the compromised system so that the malware resumes its infection procedure. Both of them are technically solvable: Lindorfer et al. proposed techniques to detect environment-sensitive malware by analyzing samples in distinct environments and identifying differences on the monitored actions so as to recognize techniques used for detection of analysis systems [20]; Balzarotti et al. proposed a system that records the system calls executed by a sample in a reference environment and replay the monitored system calls in an emulator to identify if the observed behavior is different [2]; we introduced a technique to analyze reboot-dependent malware in this paper. Other anti-analysis techniques require solutions such as to remove fingerprints from the monitoring environment, and to protect the internal analysis component from malware interaction using rootkit-inspired features, but they are out of the scope of this paper.

6 Conclusions and Future Work

In this paper, we introduced a dynamic analysis that monitors marginalized malicious programs, i.e., those malware that has not being currently addressed by publicly available systems. We discussed the challenges faced to implement this analysis system for 64-bits modern Windows operating systems due to their novel security features, as well as presented details about our proposed architecture. We evaluated our system with thousands of actual malware (among PE, CPL and .NET) to show that it is able to analyze PE+, CPL,

.NET/Mono and reboot-dependent malware. The analysis of resulting logs allowed us to observe suspicious behaviors from these samples execution, as well as to confirm the expected behavior of samples reported in the community. The features implemented in our system increase the range of malware that can be analyzed, expanding it beyond the ability of current systems. These analysis results also allow the gathering of additional information about infections by security researchers and analysts, incident response professionals, and interested users. We are currently working on making our system available to the public, and on anti-anti-analysis techniques to improve the quality of our analysis environment. An extension of this work would be the comparison of samples' behavior while running at the same time (and with the same inputs/parameters) on hypervisor-based or baremetal analysis systems. This way, it would be possible to identify evasive behavior due to the execution in emulated environments. In addition, we aim to verify our environment's resistance against known fingerprint attacks, such as those implemented on *pafish*[9].

Acknowledgements This work was supported by the Brazilian National Counsel of Technological and Scientific Development (CNPq, Universal 14/2014, process 444487/2014-0) and the Coordination for the Improvement of Higher Education Personnel (CAPES, Project FORTE, Forensics Sciences Program 24/2014, process 23038.007604/2014-69).

References

1. Afonso, V., Filho, D., Gregio, A., de Geus, P., Jino, M.: A hybrid framework to analyze web and os malware. In: Communications (ICC), 2012 IEEE International Conference on, pp. 966–970 (2012). DOI 10.1109/ICC.2012.6364108
2. Balzarotti, D., Cova, M., Karlberger, C., Kirda, E., Kruegel, C., Vigna, G.: Efficient detection of split personalities in malware. In: NDSS (2010)
3. Bayer, U., Kruegel, C., Kirda, E.: Ttanalyze: A tool for analyzing malware. In: 15th European Institute for Computer Antivirus Research Annual Conf. (2006)
4. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05, pp. 41–41. USENIX Association, Berkeley, CA, USA (2005). URL http://dl.acm.org/citation.cfm?id=1247360.1247401
5. Blog, S.L.: The inevitable mode - 64-bit zeus enhanced with tor (2013). http://securelist.com/blog/events/58184/
6. Corregedor, M., Von Solms, S.: Windows 8 32 bit - improved security? In: AFRICON, 2013, pp. 1–5. IEEE (2013)
7. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: Malware analysis via hardware virtualization extensions. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS '08, pp. 51–62. ACM, New York, NY, USA (2008). DOI 10.1145/1455770.1455779. URL http://doi.acm.org/10.1145/1455770.1455779
8. Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., Lee, W.: Virtuoso: Narrowing the semantic gap in virtual machine introspection. In: Proceedings of the 2011 IEEE Symposium on Security and Privacy, SP '11, pp. 297–312. IEEE Computer Society, Washington, DC, USA (2011). DOI 10.1109/SP.2011.11. URL http://dx.doi.org/10.1109/SP.2011.11
9. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys **44**(2), 6 (2012)
10. Fattori, A., Paleari, R., Martignoni, L., Monga, M.: Dynamic and transparent analysis of commodity production systems. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE '10, pp. 417–426. ACM, New York, NY, USA (2010). DOI 10.1145/1858996.1859085. URL http://doi.acm.org/10.1145/1858996.1859085
11. Guarnieri, C.: Cuckoo sandbox. http://www.cuckoosandbox.org/ (2013)
12. Guri, M., Kedma, G., Sela, T., Carmeli, B., Rosner, A., Elovici, Y.: Noninvasive detection of anti-forensic malware. In: Malicious and Unwanted Software: "The Americas" (MALWARE), 8th International Conference on, pp. 1–10 (2013). DOI 10.1109/MALWARE.2013.6703679
13. j00ru: Defeating windows driver signature enforcement 3: The ultimate encounter. http://j00ru.vexillium.org/?p=1455
14. Kaspersky: Equation group: Questions and answers. http://securelist.com/files/2015/02/Equation_group_questions_and_answers.pdf
15. Kirat, D., Vigna, G., Kruegel, C.: Barebox: efficient malware analysis on bare-metal. In: Proceedings of the 27th Annual Computer Security Applications Conference, pp. 403–412. ACM (2011)
16. Kirat, D., Vigna, G., Kruegel, C.: Barecloud: Bare-metal analysis-based evasive malware detection. In: 23rd USENIX Security Symposium (USENIX Security 14), pp. 287–301. USENIX Association, San Diego, CA (2014). URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kirat
17. Kruegel, C.: Full system emulation: Achieving successful automated dynamic analysis of evasive malware. https://www.blackhat.com/docs/us-14/materials/us-14-Kruegel-Full-System-Emulation-Achieving-Successful-Automated-Dynamic-Analysis-Of-Evasive-Malware.pdf (2014)
18. Lab, K.: The regin platform - nation-state ownage of gsm networks. http://securelist.com/files/2014/11/Kaspersky_Lab_whitepaper_Regin_platform_eng.pdf
19. Lindorfer, M., Di Federico, A., Maggi, F., Comparetti, P.M., Zanero, S.: Lines of malicious code: Insights into the malicious software industry. In: Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12, pp. 349–358. ACM, New York, NY, USA (2012)
20. Lindorfer, M., Kolbitsch, C., Milani Comparetti, P.: Detecting Environment-Sensitive Malware. In: Recent Advances in Intrusion Detection Symposium (2011)
21. Mercês, F.: Cpl malware - malicious control panel items. http://www.trendmicro.com/cloud-content/us/

[9] https://github.com/a0rtega/pafish

pdfs/security-intelligence/white-papers/wp-cpl-malware.pdf
22. Microsoft: Device input and output control (ioctl). https://msdn.microsoft.com/pt-br/library/windows/desktop/aa363219%28v=vs.85%29.aspx
23. Microsoft: I/o request packets. https://msdn.microsoft.com/en-us/library/windows/hardware/hh439638%28v=vs.85%29.aspx
24. Microsoft: Queueuserapc function. [https://msdn.microsoft.com/en-us/library/windows/desktop/ms684954%28v=vs.85%29.aspx
25. Microsoft: Reg_notify_class enumeration. https://msdn.microsoft.com/pt-br/library/windows/hardware/ff560950%28v=vs.85%29.aspx
26. Microsoft: Running 32-bit applications. https://msdn.microsoft.com/en-us/library/windows/desktop/aa384249%28v=vs.85%29.aspx
27. Microsoft: Trojan:win32/jorik.c. http://www.microsoft.com/security/portal/threat/encyclopedia/Entry.aspx?Name=Trojan:Win32/Jorik.C
28. Microsoft: Using cplapplet. https://msdn.microsoft.com/en-us/library/windows/desktop/cc144199%28v=vs.85%29.aspx
29. Microsoft: Win32/wootbot. http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?name=Win32%2FWootbot
30. Microsoft: CreateRemoteThread. http://msdn.microsoft.com/en-us/library/windows/desktop/ms682437(v=vs.85).aspx (2013)
31. Microsoft: CmRegisterCallback. http://msdn.microsoft.com/en-us/library/windows/hardware/ff541918(v=vs.85).aspx (2014)
32. Microsoft: CmRegisterCallbackEx. http://msdn.microsoft.com/en-us/library/windows/hardware/ff541921(v=vs.85).aspx (2014)
33. More, A., Tapaswi, S.: Virtual machine introspection: towards bridging the semantic gap. Journal of Cloud Computing **3**(1), 1–14 (2014). DOI 10.1186/s13677-014-0016-2. URL http://dx.doi.org/10.1186/s13677-014-0016-2
34. Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: Rage against the virtual machine: Hindering dynamic analysis of android malware. In: Proceedings of the Seventh European Workshop on System Security, EuroSec '14, pp. 5:1–5:6. ACM, New York, NY, USA (2014)
35. Pietrek, M.: Peering inside the pe: A tour of the win32 portable executable file format. https://msdn.microsoft.com/en-us/library/ms809762.aspx
36. Reloaded, P.: Skywing. http://uninformed.org/?v=8&a=5
37. Rienhardt, F.: Kernel-basedmonitoringonwindows(32/64bit). http://www.bitnuts.de/KernelBasedMonitoring.pdf (2012)
38. Rodionov, E., Matrosov, A.: The evolution of tdl: Conquering x64. http://www.eset.com/us/resources/white-papers/The_Evolution_of_TDL.pdf
39. Seifert, C., Steenson, R., Welch, I., Komisarczuk, P., Endicott-Popovsky, B.: Capture - a behavioral analysis tool for applications and documents. Digital Investigation **4S**, 23–30 (2007)
40. Sikorski, M., Honig, A.: Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, San Francisco, CA, USA (2012)
41. skape, Skywing: Bypassing patchguard on windows x64. http://uninformed.org/index.cgi?v=3&a=3
42. Skywing: Subverting patchguard version 2. http://www.uninformed.org/?a=1&t=txt&v=6
43. Thomas, S., Sherly, K., Dija, S.: Extraction of memory forensic artifacts from windows 7 ram image. In: Information & Communication Technologies (ICT), 2013 IEEE Conference on, pp. 937–942. IEEE (2013)
44. TrendMicro: Darkkomet. http://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/DARKCOMET
45. TrendMicro: Tspy64_zbot.aanp. http://about-threats.trendmicro.com/Malware.aspx?language=au&name=TSPY64_ZBOT.AANP
46. Willems, C., Holz, T., Freiling, F.: Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy **5**, 32–39 (2007)
47. Willems, C., Hund, R., Holz, T.: Cxpinspector: Hypervisor-based, hardware-assisted system monitoring. Tech. Rep. TR-HGI-2012-002, HGI, Ruhr-Universitat Bochum (2012)