

available at www.sciencedirect.comjournal homepage: www.elsevier.com/locate/diin
**Digital
Investigation**

Capture – A behavioral analysis tool for applications and documents

Christian Seifert^{a,*}, Ramon Steenson^a, Ian Welch^a, Peter Komisarczuk^a,
Barbara Endicott-Popovsky^b

^aSchool of Mathematics, Statistics and Computer Science – Te Kura Tatau, Victoria University of Wellington – Te Whare Wānanga o te Ūpoko o te Ika a Māui, P.O. Box 600, Wellington 6140, New Zealand

^bThe Information School, University of Washington, Box 352840, Seattle, WA 98195-2840, USA

ABSTRACT

Keywords:
Security
Forensics
Behavioral analysis
Application analysis
Document analysis

In this paper, we present Capture, a tool for behavioral analysis of applications for the Win32 operating system family. Capture is able to monitor the state of a system during the execution of applications and processing of documents, which provides the analyst with insights on how the software operates even if no source code is available. Capture differs from existing behavioral analysis tools in its ability to monitor state changes on a low kernel level and its ability to be easily used across operating systems, various versions and configurations. Capture provides a powerful mechanism to exclude event noise that naturally occurs on an idle system or when using a specific application. This mechanism is fine-grained and allows an analyst to take into account the process that causes the various state changes. As a result, this mechanism even allows Capture to analyze the behavior of documents that execute within the context of an application. We demonstrate Capture's capabilities by analyzing a malicious Microsoft Word document.

© 2007 DFRWS. Published by Elsevier Ltd. All rights reserved.

1. Introduction

Behavioral analysis is the process of reverse engineering the inner workings of applications by examining their effects on the system they operate in. Because source code is not always readily available, behavioral analysis is the major technique for determining what an application does and how it manipulates data.

Behavioral analysis has long been used in the area of computer security. Forrest et al. (1996) created system call fingerprints of applications during normal operation for their anomaly based intrusion detection system. These fingerprints allowed them to detect attacks on these applications if

abnormal system call sequences were identified. Willems et al. (2007) use behavioral analysis for automated analysis of malware in a sandboxed environment. The generated reports greatly simplify and automate the malware analysis task and Symantec (2006) uses behavioral analysis for generation of heuristics of malware. The resulting reports are used to generate heuristic detection mechanisms to identify the analyzed malware based on its behavioral characteristics.

In this paper, we present Capture – a behavioral analysis tool for applications and documents. In Section 2, we describe the difficulties behavioral analysis tools are faced with followed by an analysis on how existing tools attempt to address these difficulties in Section 3. In Section 4, we present our

* Corresponding author.

E-mail addresses: christian.seifert@mcs.vuw.ac.nz (C. Seifert), ramon.steenson@mcs.vuw.ac.nz (R. Steenson), ian.welch@mcs.vuw.ac.nz (I. Welch), peter.komisarczuk@mcs.vuw.ac.nz (P. Komisarczuk), endicott@u.washington.edu (B. Endicott-Popovsky).
1742-2876/\$ – see front matter © 2007 DFRWS. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2007.06.003

solution, the Capture tool and we describe its functionality and technical aspects. And in Section 5, we present an example of how Capture was used to analyze a malicious Microsoft Word document. Last we conclude in Section 6.

2. Problem

To determine the behavior of software on a complex operating system is a difficult endeavor. Many system events occur even when an operating system is idle. Thousands of events are generated that would overwhelm an analyst if one would simply “listen” to all events. On a clean, idle Windows XP SP2 installation, we observed 530 registry events and 60 file events within 1 min. Even if an operator focuses on state change events only, an overwhelming number of events are received: log files are written, files are adjusted for optimization purposes, etc. As processing components are added, such as opening a document with an application, this noise increases and it is difficult to decipher which events have been generated by the application as part of its normal operation and which events have been caused by a document executing within the client application. A tool that allows an analyst to observe system state, therefore, needs to be able to report on system events that are exclusively caused by the software being observed. Because the analyst works on different environments and configurations, the tool should be portable.

Once a tool captures the behavior of the software, there should be a high level of confidence that the report it generates is correct. Malware, in particular, has a tendency to take measures that avoid analysis. It, for example, encrypts its binary form to prevent static analysis. Behavioral analysis is a technique that is harder to foil, but it is possible. If an application manipulates the system using low level function calls, behavioral analysis tools monitor high level function calls which will fail to detect them and will not truly capture the behavior of the software. As a result, the monitoring of state changes should occur on the lowest possible level.

Further, it is desirable that the tool itself has a level of transparency that reveals its inner workings. Documentation usually provides one level of transparency, but it needs to be verifiable through inspection of the source code. As such, an analysis tool should also provide its source code so that it may be inspected, ensuring that the advertised functionality is actually implemented by the tool on all levels.

We have reviewed a number of tools that can be used to perform behavioral analysis: Sunbelt’s CWSandbox (Sunbelt, 2006), Microsoft’s Sysinternals (the old implementation as well as Microsoft’s rewrite) (Microsoft Corporation, 2006a), and the open-source software Winpooch (Blanchon, 2004). Some of these tools have a different primary purpose, for example Winpooch is designed to prevent malware infections, but all share the functionality of capturing behavior of an application and acting upon that information. As a behavioral analysis tool, they do not fulfill the requirements of portability, high confidence in the generated report, and transparency as shown in Fig. 1. Either their state inspection mechanism is not low level and therefore does not provide the necessary confidence in the generated analysis report or the tools do not provide the necessary portability due to technical or

Tool	Confidence in Report	Portability	Transparency
CWSandbox		✓	
Old Sysinternals	✓		✓
Rewritten Sysinternals	✓		
Winpooch	✓		✓

Fig. 1 – Comparison matrix of fulfilled requirements.

functional aspects of the specific implementation. In Section 3, we review the various implementation techniques.

3. Background

In this section, we review the implementation aspects of analysis tools. We review several techniques on state monitoring and aspects of portability that are used by behavioral analysis tools today on the Win32 platform.

3.1. State monitoring techniques

First, we review three event-based techniques that allow behavioral tools to monitor system state changes: user level API hooking, kernel level API hooking, and kernel level callbacks.

3.1.1. User level API hooking

User level API hooking is a technique that injects monitoring code around shared, for example Win32 API, function calls that an application utilizes (Ivanov, 2002). It changes the pointers in a process address space that point to functions and adjusts them to point to a user defined hook. The user defined hook can manipulate and monitor the data. This technique only injects hooking code into the application that is being executed, so no exclusion lists are necessary, because the view of the monitoring code is naturally restricted to an application of focus. This has some drawbacks. In particular, applications that directly call the kernel and avoid using the Win32 API cannot be monitored. This might be uncommon for applications to do, but would reduce confidence in the generated report because one cannot discern whether such calls have or have not occurred by examining the behavioral analysis report.

3.1.2. Kernel level API hooking

Kernel level API hooking uses a similar approach. However, instead of injecting code into an application, kernel level API hooking injects code into the kernel itself providing a system wide view of state changes that is more difficult to circumvent (Bassov, 2006). Modifications of the kernel in this manner are now being discouraged by the vendor, as they can cause other programs to crash or perform unexpectedly. Kernel level API hooking is not portable across different versions of the operating system as hooked function calls are not a supported interface and therefore are changing with each version.

3.1.3. Kernel callbacks

Instead of patching the kernel with kernel level API hooking, Microsoft encourages the usage of callback functions (Field, 2006). Callback functions are publicly supported interfaces on the kernel level that notify an application about state changes on the system. These callbacks are designed with reliability and long-term supportability in mind allowing a monitoring application to run on various versions of the Microsoft Windows operation system without modification.

3.2. Portability

Second, we review implementation options around portability.

3.2.1. Code portability

As we have already mentioned above, certain state monitoring techniques prevent portability of code across different Win32 environments and configurations. While user level API hooking is portable, kernel API hooking is not. If low level system monitoring is required, the kernel level callback mechanism is the only portable solution.

3.2.2. Filtering mechanism

The monitoring application needs to be able to filter events that occur naturally on the system so as not to overwhelm the analyst with irrelevant events. The user level API hooking has this sort of functionality already built in through its focus on application events. System wide state monitoring as being done by the kernel level technique requires a mechanism to filter events after the events have been received by the monitoring application. An exclusion list or filter needs to be configurable and should be portable across instances of the application. One should be able to import existing filters and export created filters, so they can be reused across systems.

Fig. 2 shows a comparison of how the various tools realize the requirements of the previous section. The three shaded columns on the right indicate the desired implementation, and no tool provides a desired state inspection mechanism as well as portability of code and filtering mechanism.

4. Solution

Capture, the tool that we have created and present in this paper, does fulfill all three requirements of high confidence in the report, portability and transparency. Capture was originally designed as an open-source high interaction client honeypot (Stenson and Seifert, 2006), but in stand-alone mode it

Tool	State Inspection Mechanism		Portability	
	User Level API Hooking	Kernel Level API Hooking or Call-backs	Code	Filtering Mechanism
CWssandbox	✓			✓
Old Sysinternals		✓		
Rewritten Sysinternals		✓	✓	
Winpooch		✓		✓

Fig. 2 – Comparison matrix of implementation aspects.

can also function as a behavioral analysis tool for software running on the Win32 family of operating systems (Microsoft Corporation, 1993) including the latest version of Windows Vista. In this section, we describe the functionality of Capture followed by a description of the technical aspects of the tool.

4.1. Functional description

Similarly to the other existing tools, Capture analyzes the state of the operating system and applications that execute on the system by monitoring the file system, the registry, and process monitor and generating reports for any events received by the three monitors. The three monitors are described below:

- The *file system monitor* captures read or write events to all mounted file systems as the events occur on the system, including information about when the event occurred, the event type (such as read and write), the process that triggered this event, and the fully qualifying name of the file or directory that was acted upon.
- The *registry monitor* captures a similar set of events, but focuses on the Windows Registry, which stores configuration options of the operating system and installed applications in a large construct of key/value pairs that are arranged in five sections, or “hives,” and hierarchically in paths similar to the file system. The registry monitor reports the time with a resolution in milliseconds, the process that triggered the registry event, the path to the key where the action occurred, and the type of action performed on the key. Since the Windows Registry allows the user to read and manipulate the key/value pairs, as well as to discover the content of a particular registry path, more event types are captured by the registry monitor, such as OpenKey, CreateKey, CloseKey, EnumerateKey, EnumerateValKey, QueryValKey, QueryKey, SetValKey, SetKey, DeleteValKey, and DeleteKey.
- The *process monitor* observes the creation and destruction of processes, but does not report on already running processes. With each event it captures the time, whether the process was created or destroyed, and the fully qualifying file name that represents the process. In addition, the process monitor captures the parent process, which assists in determining which process created or destroyed the process that caused the event to trigger. For example, when one double clicks on an executable in Windows Explorer, Explorer is the parent process.

Since normal events are constantly generated, *portable* exclusion lists instruct the monitors to omit events from the final report. There is one exclusion list for each monitor: FileSystemMonitor.exl, RegistryMonitor.exl, and ProcessMonitor.exl. The exclusion lists are simple text based files that can be created once and moved around different environments and configurations. This allows the analyst community to create a set of reusable exclusion lists that can be shared. For example, one could create an exclusion list for an idle Microsoft Windows XP SP2 system. Analysts can reuse this list and customize it for their specific needs.

The default policy of the exclusion list is to report all events. Each row within the exclusion list allows specification

```
# [Evt Type] [Process Name] [File Path]
+ Read      .*          .*
+ Write     .*          C:\WIN\.*
- Write     .*          C:\WIN\sys\.*
+ Write     C:\BROWSER.EXE C:\CACHE\.*
```

Fig. 3 – Example of file exclusion list.

of an exclusion list rule. A user can specify omission or inclusion of events by event type and the object name. An omission is denoted by a plus and an explicit inclusion by a minus sign at the beginning of the rule. This allows one to omit a larger group of events and then overwrite these settings for a subset of events to be included. The exclusion list is processed in sequence to determine the final rules that should be applied. Fig. 3 shows an example of a file monitor exclusion list. Lines 2 and 3 express that write events in C:\WIN\.* are omitted, but that events in C:\WIN\sys\.* are reported.

By allowing the user to specify the process that caused the event, the exclusion list is a powerful mechanism for partitioning the monitoring rules further. For example, one could specify that BROWSER.EXE's write access to its cache is not recorded, but access to it by any other application is, as shown in line 4.

Each not-excluded event that is triggered during the execution of Capture is output into a report. The report includes the name of the monitor and the event information described above. The report is a simple comma separated value report. An example of a report is shown in Fig. 4, which shows process, registry, and file system events.

4.2. Technical description

Capture consists of two components, a set of kernel drivers and a user space process (Fig. 5). The kernel drivers operate in kernel space and use event-based detection mechanisms for monitoring the system's state changes that application like Microsoft Word and Internet Explorer cause. The user space process, which communicates with the kernel drivers, filters the events based on the exclusion lists and outputs the events into a report. Each component is written in unmanaged C code, code that directly compiles into machine code, and described in the following sections.

4.2.1. Kernel drivers

The Capture client uses kernel drivers to monitor the system by using the existing kernel callback mechanism of the kernel that notifies registered drivers when a certain event happens. These callbacks invoke functions inside of a kernel driver and pass the actual event information so that it can either be modified or, in Capture's case, monitored. The following callback functions are registered by Capture:

- CmRegistryCallback
- PsSetCreateProcessNotifyRoutine
- FilterLoad, FltRegisterFilter

```
"Timestamp","Monitor","EventType","Causator Process","Event Fully Qualifying Name"
"22/02/2007 18:00:01.637","F","W"C:\WIN\sys\services.exe", "C:\WIN\sys\conf\SecEvt.Evt"
"22/02/2007 18:00:01.899","P","C","C:\WIN\explorer.exe", "C:\WIN\sys\calc.exe"
"22/02/2007 18:00:01.111","R","SetValKey", "C:\WIN\sys\calc.exe", "HKCU\Software\MS\Calc\t1"
```

Fig. 4 – Example of report output.

The CmRegistryCallback function allows the registry monitor driver to observe the Windows Registry. The Windows Registry Manager invokes this function each time the registry is modified. Since create and delete events can occur, the callback function is invoked before and after the event is committed to the registry, which allows Capture also to monitor delete and creation events on the registry. The event contains information such as the action that occurred on the registry, the process that asked for this event to be carried out, as well as a path to the event, such as a registry key.

The PsSetCreateProcessNotifyRoutine function allows the process monitor driver to watch changes to running processes. The Windows Process Manager invokes it each time a process is created or terminated. The event data specify what occurred to a process and the name of the actual process that was operated on. It also contains the parent process that requested the action to occur.

The last kernel driver, the file monitor driver, works slightly differently. Instead of registering with a callback mechanism on the kernel to receive events directly about the file system, the file monitor driver is a minifilter driver that sits between the I/O manager of the Windows kernel and the base file system. A Windows Filter Manager manages these drivers and registers the callbacks to be called when a particular file event happens. Capture uses the FilterLoad function to load the file monitor minifilter driver with the Filter Manager. After it has been loaded, the minifilter driver proceeds to register with the Filter Manager on what events it wants to listen to. It is then notified when a file is either read or written to. Again, information about the process that caused the event and the path of the file is provided with the event.

When events are received inside the Capture kernel drivers, they are queued waiting to be sent to the user space component of the client. This is accomplished by passing a user allocated buffer from user space into kernel space where the kernel drivers then copy information into that buffer, so the application can process it in user space.

4.2.2. User space process

The Capture process is an application that resides in user space. It is responsible to load the drivers, process the events received by the drivers and output the events to the report.

As mentioned above, the user space application, once it has loaded the drivers, creates a buffer and passes it from user space to the kernel drivers. Passing of the buffer occurs via the Win32 API and the I/O Manager. The kernel drivers copy the event data into the buffer, so the user level application can process the events. Each event is serialized and compared against the entries in the exclusion list. The exclusion lists are built using regular expressions, which means event exclusions can be grouped into one line. This functionality is provided by the Boost::regex library. For each monitor, an exclusion list is parsed and internally mapped between event types and allowed regular expressions. If a received event is

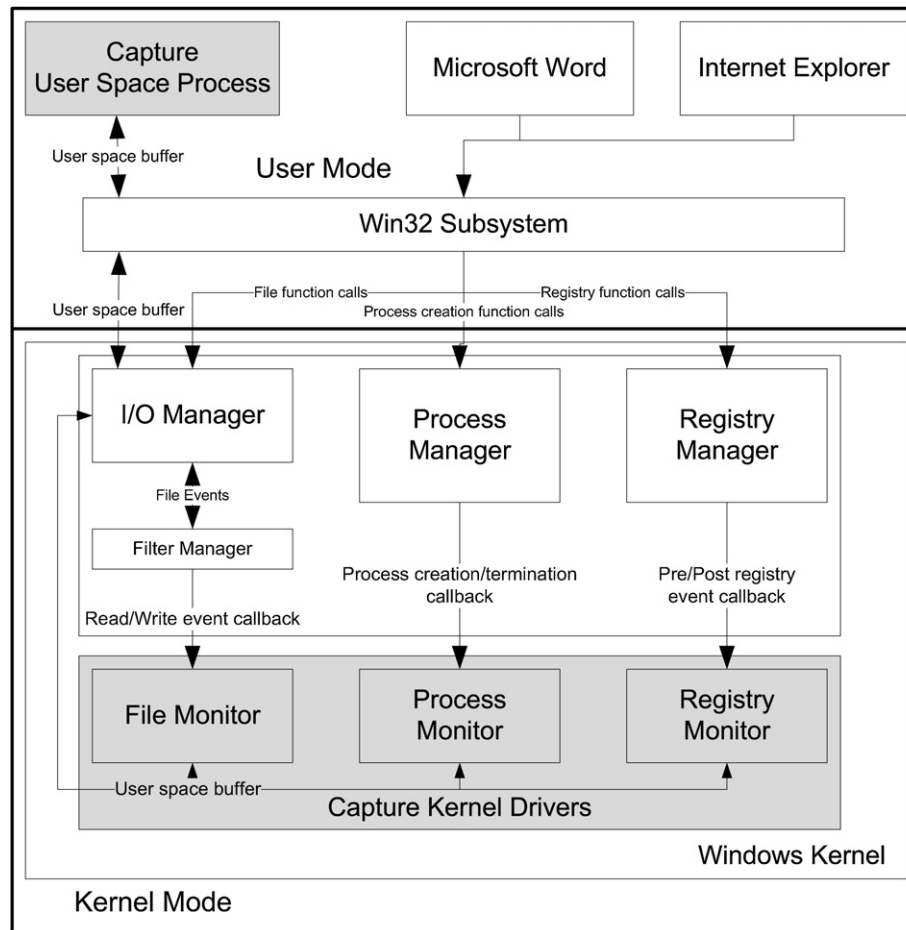


Fig. 5 – Capture architecture diagram.

included in the list, the event is dropped; otherwise, it is output to the final report that Capture generates.

5. Results

To demonstrate the capabilities of Capture, we expose Capture to a malicious Microsoft Word document that exploits vulnerability MS06-027 (Microsoft Corporation, 2006b). We have chosen to analyze an Office document because it represents a more complex case in application analysis than the mere analysis of a stand-alone executable. The Word document will be executed within a legitimate application and the analysis tool, as a result, is tasked to differentiate between legitimate actions of the application and actions performed by the malware document. This situation allows us to show the power of Capture's exclusion list and its system wide monitoring mechanism.

The analysis is performed on a virtualized environment that runs an unpatched Microsoft Windows XP SP2 installation with Microsoft Word 2003. Capture has been installed on this machine and configured to exclude state changes that occur during idle operation of the system, such as addition to log files, as well as during the legitimate operation of the Microsoft Word application. These excluded events are events that revolve around keeping backup copies of

documents, creation of document history, etc. The corresponding exclusion lists are shown in Fig. 6 (the Microsoft Windows XP SP2 exclusion list has been omitted because of space limitations). With these exclusion lists, no events will be reported when opening or closing an existing benign Word document.

With this configuration, we proceed to open the malicious Word document *a1.do* by double clicking on the document from Windows Explorer. As expected, Word starts and attempts to open the document, but shortly after our action, Word crashes. After 30 s, we proceed to inspect the report produced by Capture. The formatted report is shown in Fig. 7. The report shows that 14 unique unauthorized state changes occurred. First, we observe that Microsoft Word writes a file *C:\WIN\sys\exe* and then proceeds to execute this newly written out file. This newly created file and process are likely to be the payload of the malicious Word document. From this point forward, the Microsoft Word application is not involved in further state changes, but rather the *.exe* payload takes over to modify the registry and write additional files to the system. Note that the information about the process that cause the state change is helpful to follow the events that occurred. Line 5 shows the malicious document we originally opened is being written to a temporary file as well as a process called *WINWORD.exe* as shown on line 6. *WINWORD.exe* is opened in line 11 and proceeds to further modify the registry and

File Exclusion List

#	[Evt Type]	[Process Name]	[File Path]
+	Write	C:\\WIN\\explorer.exe	C:\\Documents and Settings\\.+\\Recent\\.+\\.lnk
+	Write	C:\\WINDOWS\\explorer.exe	C:\\Documents and Settings\\.+\\Start Menu\\Programs\\Microsoft Office\\ Microsoft Office Word 2003.+
+	Write	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	C:\\Documents and Settings\\.+\\Application Data\\Microsoft\\Office\\Word11\\.pip
+	Write	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	C:\\Documents and Settings\\.+\\Application Data\\Microsoft\\Office\\Recent\\.+
+	Write	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	C:\\Documents and Settings\\.+\\Application Data\\Microsoft\\Templates\\~\\\$Normal.dot
+	Write	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	C:\\Documents and Settings\\.+\\Application Data\\Microsoft\\Proof\\~\\\$CUSTOM.DIC
+	Write	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	C:\\Documents and Settings\\.+\\Application Data\\Microsoft\\Proof\\CUSTOM.DIC

Process Exclusion List

#	[Process Created]	[Parent Process]	[Process Path]
+	WINWORD.EXE	.*	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE

Registry Exclusion List

#	[Evt Type]	[Process Name]	[Registry Path]
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKCU\\Software\\Microsoft\\Windows\\ShellNoRoam\\.+
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer.+
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKCU\\Software\\Microsoft\\Office\\Common\\.+
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKCU\\Software\\Microsoft\\Office\\11.0\\Common\\.+
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKCU\\Software\\Microsoft\\Office\\11.0\\Word\\.+
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKLM\\SOFTWARE\\Microsoft\\Cryptography\\RNG\\Seed
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Installer\\UserData\\.+
+	SetVal Key	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer.+
+	Delete ValKey	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKCU\\Software\\Microsoft\\Office\\Common\\.+
+	Delete ValKey	C:\\Program Files\\Microsoft Office\\OFFICE11\\WINWORD.EXE	HKCU\\Software\\Microsoft\\Office\\11.0\\Word\\.+

Fig. 6 – Microsoft Word exclusion lists.

file system. It is expected that WINWORD.exe inserts itself whenever the Microsoft Word application is executed, allowing the malware to execute without raising suspicion. Whether it exhibits viral behavior and replicates itself via newly created documents or whether it exhibits spyware qualities to collect information about the user and system remains to be determined in further analysis, which is beyond the scope of this paper.

As shown, Capture has successfully been used to determine the behavior of a malicious document. While further manual analysis remains to be undertaken, the tool allowed us to quickly assess whether the document was indeed malicious. Such analysis could be done in an automated fashion across a set of applications and documents. Capture also conveyed information about the state changes that occurred on the system. Because the system is now contaminated with malware, an analyst would have to proceed to an offline

analysis, but with the information provided in the report, a good foundation has been laid for a speedy and comprehensive offline analysis.

6. Conclusion

In this paper, we have presented Capture, an open-source tool for behavioral analysis of software. We have presented the functionality and technical details of this tool that fulfill the needs of the analyst: (1) system state monitoring with high confidence in the generated reports, (2) portability with a future proof state monitoring technique and portable exclusion lists, and (3) transparency through an open-source approach.

We presented a powerful *portable* exclusion list mechanism that allows to omit noise that occurs naturally on a system. With the power of regular expressions, the analysis is able

Time	Mon	Event	Causator Process	Event Fully Qualifying Name
05/04/2007 17:02:01.131	F	Write	C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE	C:\WIN\sys\exe
05/04/2007 17:02:01.200	P	Create	C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE	C:\WIN\sys\exe
05/04/2007 17:02:01.320	R	SetVal Key	C:\WIN\sys\exe	HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed
05/04/2007 17:02:01.440	R	SetVal Key	C:\WIN\sys\exe	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Personal
05/04/2007 17:02:01.531	F	Write	C:\WIN\sys\exe	C:\Documents and Settings\User\Local Settings\Temp\al.do
05/04/2007 17:02:01.680	F	Write	C:\WIN\sys\exe	C:\Documents and Settings\User\Local Settings\Temp\WINWORD.EXE
05/04/2007 17:02:01.681	R	SetVal Key	C:\WIN\sys\exe	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\{key}\BaseClass
05/04/2007 17:02:01.721	R	SetVal Key	C:\WIN\sys\exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\CommonDo
05/04/2007 17:02:01.731	R	SetVal Key	C:\WIN\sys\exe	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Desktop
05/04/2007 17:02:01.888	R	SetVal Key	C:\WIN\sys\exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\CommonDe
05/04/2007 17:02:01.982	P	Create	C:\WIN\sys\exe	C:\DOCUME~1\USER\LOCALS~1\Temp\WINWORD.EXE
05/04/2007 17:02:02.544	R	SetVal Key	C:\DOCUME~1\CHRIST~1\LOCALS~1\Temp\WINWORD.EXE	HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed
05/04/2007 17:02:02.639	R	Delete ValKey	C:\DOCUME~1\CHRIST~1\LOCALS~1\Temp\WINWORD.EXE	HKLM\SOFTWARE\Microsoft\PCHealth\Error Reporting\DW\DWFileTreeRoot
05/04/2007 17:02:02.721	F	Write	C:\DOCUME~1\CHRIST~1\LOCALS~1\Temp\WINWORD.EXE	C:\Documents and Settings\User\Local Settings\Temp\3668_appcompa.txt

Fig. 7 – Capture report on execution of malicious Word document.

to exclude groups of events on one line of the exclusion lists. In addition, the mechanism is fine-grained allowing the analyst to exclude events on a process level. For example, we have illustrated that it is possible to report on write events to a browser cache by any application but the browser itself.

In Section 5, we have presented the feasibility of using Capture as a tool to analyze the behavior of a document executing within the context of an application. We have opened a malicious Microsoft Word file and observed its behavior on the system. This example shows the power of the exclusion list that assists in the analysis. While we opened a document with the client application, Microsoft Word, we were not overwhelmed by thousands of normal system events that occur upon such action. Instead, the exclusion list allowed us to concentrate on examining the behavior of the malicious document.

In future versions of the tool, we are planning to add additional capabilities around the level of information captured. Currently, Capture only obtains identifying event information, such as the name of the underlying object and the action that was taken on the object. We are planning to capture the content of the state change itself, such as the registry key value before and after the modification, and store it for later comparison. In addition, we are planning to extend the tool around system state monitoring, such as monitoring of network connections, that the tool currently does not support.

Capture relies on the kernel callback functions for its information. There is the possibility for a malicious application to modify the kernel and change the functionality of these callbacks. We do not want to put the burden on Capture to determine whether the kernel is intact, because existing tools already allow such an assessment. However, Capture should have the capability to determine whether the kernel was modified during the operation of the software that one would like to analyze. A kernel integrity monitor would provide such functionality and is a top priority for the next version of the tool.

This current version of Capture can easily be detected by software that might be running on the system. We are not planning to introduce stealth capabilities to Capture, because it would place Capture into the rootkit family necessitating implementation techniques that are not portable, namely kernel level API hooking. However, we are intending to implement capabilities that allow Capture to assess whether any software attempts to determine Capture's existence. If no such attempts were made, one can be assured that the behavior of the application was not altered because of the presence of the Capture tool.

The Capture tool is open-source and available from our web site at: <http://www.nz-honeynet.org/capture-standalone.html>.

Acknowledgements

We would like to thank Manuel Fries from Rising Antivirus, Ltd. in providing us with a malware sample that we used in our sample analysis.

REFERENCES

- Blanchon B. Winpooch watchdog. Available from: <<http://winpooch.free.fr>>; 2004 [accessed 15.03.07].
- Bassov A. Hooking the kernel directly. Available from: <http://www.codeproject.com/system/soviet_direct_hooking.asp>; 2006 [accessed 10.11.06].
- Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA. A sense of self for Unix processes. In: 1996 IEEE symposium on security and privacy. Oakland: IEEE; 1996. p. 120-8.
- Field S. An introduction to kernel patch protection. Available from: <<http://blogs.msdn.com/windowsvistasecurity/archive/2006/08/11/695993.aspx>>; 2006 [accessed 15.03.07].

Ivanov I. API hooking revealed. Available from: <<http://www.codeproject.com/system/hooksys.asp>>; 2002 [accessed 13.03.06].

Microsoft Corporation. Windows Sysinternals. Available from: <<http://www.microsoft.com/technet/sysinternals/default.msp>>; 2006 [accessed 15.03.07].

Microsoft Corporation. Windows (32bit) home page; 1993.

Microsoft Corporation. Microsoft security bulletin MS06-027: vulnerability in Microsoft Word could allow remote code execution. Available from: <<http://www.microsoft.com/technet/security/bulletin/ms06-027.msp>>; 2006 [accessed 05.04.07].

Symantec. Understanding heuristics: symantec's bloodhound technology. Available from: <<http://www.symantec.com/avcenter/reference/heuristc.pdf>>; 1997 [accessed 25.06.06].

Sunbelt Software USA. Sunbelt CWSandbox. Available from: <<http://www.sunbelt-software.com/Developer/Sunbelt-CWSandbox/>>; 2006 [accessed 15.03.07].

Steenson R, Seifert C. Capture – honeypot client. Available from: <<http://www.nz-honeynet.org/capture.html>>; 2006 [accessed 22.02.07].

Willems C, Holz T, Freiling F. Toward automated dynamic malware analysis using CWSandbox. IEEE Secur Priv 2007;5(2):32–9.

Christian Seifert is a PhD candidate at Victoria University of Wellington, New Zealand. He also leads the New Zealand Honeynet Alliance, member organization of the internationally operating Honeynet Research Alliance that uses honeypots to attract, track and study current and emerging security threats across the Internet. Christian has an MS in Software Engineering with a focus on computer security from Seattle University, WA. His research interests include network security, honeypot technology, in particular client honeypot technology, and malware analysis.

Ramon Steenson is a software engineer working as a research assistant at the University of Victoria in Wellington, New Zealand. He has a Bachelor of Information Technology from that University. His interests include honeypot technologies, computer security, and kernel development.

Ian Welch is currently a lecturer in the School of Mathematics, Statistics and Computer Science at the University of Victoria at Wellington, New Zealand. He has a PhD from the University of Newcastle upon Tyne (UK). His research interests include intrusion detection, privacy-preserving auctions, community informatics and the preservation of New Zealand-developed computer games.

Peter Komisarczuk received his PhD from the University of Surrey (UK) and is a senior lecturer at the University of Victoria in Wellington New Zealand. He teaches Networking, Internet Technology, Computer Architecture and coordinates student work experience. His research interests include next generation networks, Internet architecture, grid computing and crosslayer and location aided techniques in wireless networking.

Barbara Endicott-Popovsky has an MS in Information Systems Engineering and an MBA. She is completing her PhD in computer science, focused on computer security/computer forensics, at the University of Idaho's NSA Center for Academic Excellence in Information Assurance. As Director for the CIAC, an emerging Center for Academic Excellence in Information Assurance, she is responsible for guiding the research direction and developing an information assurance curriculum. She has a joint faculty appointment with the Information School and the Computer Science Department at the University of Washington at Tacoma. Her research interests include forensic-ready networks and integrating secure coding practices.