СЕКЦИЯ 2
СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ И КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ
Подсекция 2.1 Вычислительная техника и автоматизированные системы управления

UDK 629.78.002.5.

CONTAINER-BASED SYSTEM FOR MALWARE ANALYSIS IN HIGH FIDELITY ENVIRONMENT

Anton Kopeikin[1], Arnur G. Tokhtabayev[2] and Nurlan Tashatov[3]

[1] MS Student, L.N. Gumilyov Eurasian National University, Kazakhstan
[2] CEO, T&T Security, Kazakhstan
[3] Scientific advisor, L.N. Gumilyov Eurasian National University, Kazakhstan

I. INTRODUCTION

In the past years, malware developers continuously have been searching for yet new ways to attack hosts and evade existing popular cyber-defense systems, e.g. anti-viruses (AV) and intrusion detection systems (IDS). To intrude, an attacker must solve at least two challenges: develop a malware that is not detected by AVs and deliver the malware to a victim host. Attackers advanced in both challenges.

To avoid detection, adversaries develop complex *zero-day malware* that is not yet known for current versions of AV. This ensures that each malware sample is unique and AV signature created for it will not match any other sample, which makes signature approach obsolete. In addition attackers tend to use so-called distributed malware by partitioning malicious functionality among several files, making each file individually benign, yet in combination they achieve malicious goal. Since modern AV hardly correlate activity of several processes, such an attack will progress undetected.

One of the recent malware distribution method widely adopted by adversaries are *user-oriented attacks*, which are directed to user errors. These attacks include spear phishing, strategic web compromise, contaminated SEO, social network malware and insider threat. In fact such attacks are often preceded by social engineering phases. As a result, the user is persuaded to ignore/overwrite alerts and recommendations of IDS. Then the user performs dangerous operations with vulnerable applications such as opening suspicious/malicious web links, executing suspicious files or opening documents with mistrustful script. In combination with zero-day exploits this leads to proliferation of malicious objects.

Due to high efficiency of these offensive approaches, they are also frequently used in professional targeted attacks against organizations or groups of people such as Advanced Persistence Threat (APT).

Unfortunately, current IDSs do not offer credible defensive solutions for these problems. It is clear, that one needs innovate solutions that must provide security even for a ignorant user.

In this paper we present a novel intrusion prevention system called a secure container that protects a host from abovementioned attacks. When necessity comes, secure container seamlessly analyzes malicious activity of vulnerable applications being under attack in a specific virtual environment that enables high fidelity in malicious activity analysis. Such high execution environment fidelity is achieved by user interaction simulation in real time. The user interaction simulator recognizes GUI components and clicks through them according to click pattern of a typical user, e.g. office worker.

Our system allows for run-time detection of malicious functionalities. To this end, we applied modified Hierarchal Colored Petri Nets for deep dynamic analysis of programs functionality.

The contributions of the paper are as follows: description of an emerging threat named user-

oriented attacks, a novel functionality detection technology, introduction and evaluation of the developed secure container system that protects from zero-day and user-oriented attacks.

To demonstrate our approach we implemented a prototype of the secure container system. The system has been tested for detection of several malicious functionalities employed by network worms and bots, including self-replication engines and various malicious payloads.

II. SYSTEM DESCRIPTION

The secure container system provides seamless malware identification at the level of program activity. The system enables object execution isolation and effective malicious activity analysis

Isolation allows for running checked malware and seamlessly imitates all user interaction with vulnerable applications in segregated, disposable containers, which are backed by virtual machines. It ensures that such applications will not harm the OS being checked under attacking scenarios.

Malicious activity analysis allows for mitigating attacks by continuously monitoring processes behavior at run-time inside each container. We employ a technology of functionality analysis based on modified CPN [14], which detects hidden and complex malicious functionality at the system call level.

A. Functionality recognition

From OS perspective, processes invoke API functions or system calls to perform system object operations (manipulations) that complete some semantically distinct *system actions*, such as writing data to a file or sending data to a specified IP address. We define individual functionality as a combination of such *system actions* that achieve a certain high-level objective.

The functionality is recognized in two stages: system calls and object manipulation (API traits). A manipulation may be performed through several alternative APIs operating on the same Kernel objects. API may invoke several additional minor system calls that are not critical for the manipulation implementation. Hence, only the essential, semantically critical part of an API should be recognized.

In our CPN models are be employed for the recognition of malicious functionalities.

A CPN could formally be defined as a tuple [2]: $CPN=(S,P,T,A,N,C,G,E,I)$ (2), where: **S** – color set, **P** – set of places, **T** – set of transitions, **A** – set of arcs, **N** – node function, **C** – color function, **G** – guard function, **E** – arc expression function, **I** – initialization function.

To recognize functionalities CPN must reflect objects and manipulations. Hence, CPN places must represent the following states: created objects, which are ready to be manipulated; manipulations on the objects; pseudo states routing the control flow and functionalities.

CPN has a *set of places* (P) that consists of four disjoint dedicated subsets – Object places, Manipulation places, Functional places and Pseudo places: $P=P_{obj} \dot{E} P_{manip} \dot{E} P_{fun} \dot{E} P_{pseudo}$, such that, each Object place is associated with a unique OS object; every Manipulation place represents a particular (individual) operation of an object; any Functional place corresponds to a unique functionality and a Pseudo place. Functional place tokens represent successful recognition of the given functionality.

Places of CP-nets represent executed object operations; therefore a transition must be attributed to execution of one of the equivalent system calls implementing the respective manipulations. The set of transitions consists of three sets: $T=T_{man} \cup T_{fun} \cup T_{pseudo}$, where T_{man} - represent system calls or a user level manipulation. T_{fun} - transitions, which constitute functionality trigger, T_{pseudo} - pseudo transitions that reflect conditional branches. Transition guard expressions check manipulation handles and parameters to ensure that transitions are enabled only by manipulations with correct attributes specified by functionality. It provides flexibility to distinguish similar yet semantically different functionalities.

III. SYSTEM EVALUATION

We experimented with various malware families. By description, the selected malware family set exposed the following malicious functionalities.

Replication engines: **R1.** *Self code injection* – a malware infects an executable file through injecting its code into the executable body and replacing code entry points; **R2.** *Download and Execute* – Downloads a file from the Internet and executes it. Used as a part of self-propagation engine of network worms [3], hence exposed by exploited processes and Trojan-downloaders; **R3.** *Remote shell* –Used as a part of propagation engine for network worms.

Malicious payloads: **P1.** *Dll/thread injection* - Injects DLL/thread to the address space of a process. Used for password stealing or process control hijacking; **P2.** *Self manage system script create and execute* – This malicious process creates and executes command script. The script implements a functionality that relocates/deletes the malware image to conceal its footprint. Afterwards, the script usually erases itself; **P3.** *Remote hook* - sets a remote hook for a particular event (keylogging).

These functionalities were specified and translated to CPNs. In order to verify the detection rate, we experimented with the malware known for performing at least one of the malicious functionalities:

file viruses (Neo, Abigor, Crucio, Savior, Nother, Halen,), network worms (Welchia, Bozori, Iberio, HLLW.Raleka, Alasro, Kassbot, Francette), bots and trojans (Zbot, SpyBot, RxBot, Banker, Lespy). We run each malware image in the corresponding environment enabling the malware to execute its payloads or replicate properly. In order to evaluate the false positive rate, we run multitude of benign software that include web-browsers, messengers, email clients, file utilities, network/system utilities and office tools. We run the tested software under various conditions to expose their functionalities.

A. Detection Results

The results of our experiments are shown in Table 1. For the legitimate software or malware samples, each cell indicates how many programs secure container based on the given functionality detected. For example, 4/4 means that there are four instances from the set that have the given functionality and all four exposed it and were detected.

False negatives (detection rate). As Table 1 indicates, for each malware family that has the given functionality, secure container successfully detected the functionality and blocked the malware from propagating into host OS.

False positives. It could be seen that, Table 1 contains several false positives (FP). Below, we give the following possible reasons of why a particular functionality was exposed by legitimate software.

1. *Executable download and execute* functionality can be performed by Internet browsers or file managers. Mostly, such activity is performed on behalf of the end-user. In addition, many programs periodically check for updates. If there is an update available, the program downloads it and then executes. This functionality can also be tagged as “download and execute”.

2. *DLL/thread injection* can be performed by user/system monitoring software. Particularly, Easy Hook library injects DLL to trace API calls invoked by an arbitrary program. WinSpy program accomplishes DLL injection in order to retrieve window objects data of a foreign program.

3. *Self manage script* was exposed by Easy Hook software which exiting functions run a *cmd* script that waits the hooking process to end, then removes the hooked DLLs.

4. *Remote hooking* functionality can be executed by chat programs to identify whether a user is idle. These programs hook into other processes for the input events such as keystroke and mouse message.

Indeed, our methodology allows for specifying and detecting any functionality. We believe that behavior-based detection of some complex malicious payloads, such as password stealing, may be most successful by utilizing a strategically chosen set of several primitive functionalities. On the other hand, secure container isolates all legitimate processes so that positives will not affect usability.

Finally, we evaluated performance of our user interaction simulator with tens of thousands of modern malware and unwanted software (such as riskware, key generators). The experiments indicated that 25% of tested malware exposed some kind of GUI with which our system interacted. This demonstrates effectiveness and necessity of our user activity simulating approach.

Table 1 Functionalities detection rate and false positive rate

•			R1	R2	R3	P1	P2	P3
Legitimate software	200	Windows system tools, office apps, other utilities				1	1	
	2	Web browsers (Opera, IE)		2				1
	2	E-mail clients (Outlook Expr, Eudora)						
	1	Instant messaging client (Yahoo messenger)		1				1
	2	File managers (FAR,Win Exp)		1				
	2	Network tools (Ping, Telnet)						
		Total detected		4/210		1/210	1/210	2/210
Malware		File viruses	✔					
		Network worm shell codes		✔	✔	✔		✔
		SpyBot.gen family		✔				✔
		Banker family		✔		✔		✔
		Zbot family		✔		✔	all	✔
False positive (%)			0	1.92	0	0.48	0.48	0.96
Detection rate (%)			100	100	100	100	100	100

B. Runtime overhead

The secure container prototype was executed in MS Windows 7 running on an Intel Core i7-3517U (2,4GHz) processor with 4 Gb of ram. We recorded overhead for three activities: web browsing (google chrome), video watching in the browser and PDF reading in Acrobat Reader. Figure 1 shows system CPU overhead imposed by these activities performed when protected by secure container (solid line) and natively, without secure container (dashed line). One can see that practically the secure container overhead is not much different from the native one. On average, secure container imposes about 7% CPU overhead versus native execution. As per memory, the system incurs only 3% overhead.

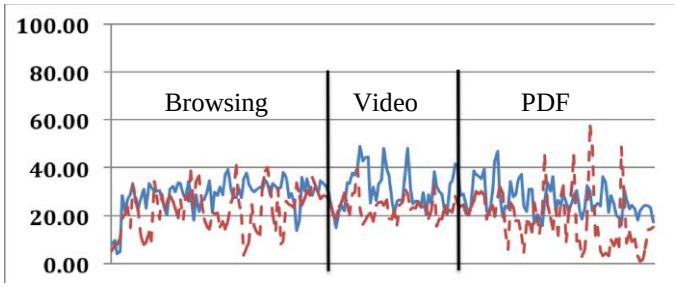

Figure 1 System overheads with/without secure container

Such a low CPU overhead could be credited to our highly efficient behavioral monitoring module. In fact that we hook only a small subset of the system calls that are part of a given modified CPN. Since the system call monitor is implemented in the Kernel mode, hooking a small subset of the system calls minimizes the number of computations needed to process by functionality detector.[1]

IV. BACKGROUND AND RELATED WORK

The secure container could be attributed to behavior-based IDS. The IDS such as [4]-[11] recognize only malicious activity in the context of a single process. Papers [4]-[6] propose tracing sequences of system calls to reveal misuse in OS objects manipulations. In contrast, our approach recognizes of complex functionalities involving interrelated sessions of object operations of multiple processes.

Works [6]-[8] target dynamic behavior analysis. Ones detect a “gene of self-replication” from object operations and activity blocks [7] but lack an efficient recognition mechanism [8]. Others

utilize so-called behavior graphs [9, 10]. Our modified CPN model represents an executed system call chain as one token residing in the corresponding place. Such token semantics allows for processing multiple system call chain instances to recognize an inter-process activity as well.

Adversaries may reduce “malicious footprint” to make the activity less suspicious in terms of behavioral statistics. They use mimicry attacks to match normality profile of IDS. Since we recognize activity on the highest semantic level, it is hardly possible to conduct a mimicry attack such that it would go unnoticed while executing certain functionality represented in our modified CPN model.

V. CONCLUSION

We introduced secure container system that enables identification of targeted and user-oriented attacks. To provide robust malware activity analysis secure container uses modified Hierarchal Colored Petri Nets for run-time recognition of malicious functionalities. The secure container provides high fidelity in malicious activity analysis, which is achieved by user interaction simulation in real time. The user interaction simulator recognizes GUI components and clicks through them according to click pattern of a typical user, e.g. office worker. Due such features secure container system is instrumental in enabling security in such modes (scenarios) when typical AV products cannot guarantee security.

We evaluated the secure container prototype with corpus of real malware families. Results showed high efficiency of secure container in detecting and blocking various malware while having low system overhead. Ultimately, secure container enabled us to securely and efficiently operate on insecure/malicious resources.

ACKNOWLEDGMENT

This research effort is funded by T&T Security LLP, Kazakhstan and is partially supported by scientific projects of L.N. Gumilyov Eurasian National University managed by the authors of this paper.

REFERENCES

[1] Cohen, F., 1987. "Computer Viruses Theory and Experiments," Computers and Security, vol. 6, pp

[2] Kurt Jensen. “Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use”, volume 1, *Springer-Verlag*, Berlin, 1996.

[3] A. G. Tokhtabayev, V. A. Skormin and A. M. Dolgikh, “Detection of Worm Propagation Engines in the System Call Domain using Colored Petri Nets ”, In Proc. *IEEE IPCCC ’07*, USA, Dec. 2008

[4] M. Bernaschi, E. Grabrielli, L. Mancini. "Operating System Enhancements to Prevent the Misuse of System Calls", in *Proc. ACM CCS 2000*, pp. 174 – 183, 2000.

[5] D. Kang, D. Fuller, and V. Honavar. “Learning classifiers for misuse and anomaly detection using a bag of system calls representation”. in *Proc. 6th IEEE Systems Man and Cybernetics Information Assurance Workshop (IAW)*, pp. 118-125, 2005.

[6] Ulrich Bayer at al., “Dynamic analysis of malicious code”, *Journal of Computer Virology*, vol. 2, no. 1, pp. 67-77, 2006.

[7] V. Skormin, A. Volynkin at al., “Run-Time Detection of Malicious Self-Replication in Binary Executables” *Journal of Computer Security*, vol. 15, no. 2, pp. 273-301, 2007.

[8] United States Patent 6973577 B1 “System and Method for Dynamically Detecting Computer Viruses Through Associative Behavioral Analysis of Runtime State”, Victor Kouznetsov, Dec 6, 2005

[9] M. Christodorescu, S. Jha and C. Kruegel, “Mining specications of malicious behavior”, In Proc. *ESEC-FSE’07, NY, USA* 2007.

[10] Lorenzo Martignoni at al., “A Layered Architecture for Detecting Malicious Behaviors”, In Proc. *RAID’08*.