Application of CLIPS Expert System to Malware Detection System

Zhou Ruili, Pan Jianfeng, Tan Xiaobin, Xi Hongsheng
Department of Automation, University of Science and Technology of China
relly@ustc.edu, pjf@ustc.edu, xbtan@ustc.edu.cn, xihs@ustc.edu.cn

Abstract

Malware detection is a crucial aspect of software security. Traditional signature-based detection method cannot detect zero-day attacks and some malware adopting some circumvention techniques such as polymorphic, metamorphic, obfuscation and packer. So some anomaly-based detection techniques are introduced to overcome this drawback, but these techniques have high false alarm rate and the complexity involved in determining what features should be learned in the training phase. In order to overcome these shortcomings, we propose a malware detection system based on expert systems in this paper. This system integrates signature-based analysis and anomaly-detection technique together. The signature is anomaly behavioral signatures. Accord to expertise about malware's major suspicious behaviors, we build the knowledge base of the expert system. And we design a behavior gathering component to intercept anomaly behaviors happened in the operating system and get significant traces leaved by malware, then present these behaviors and traces as facts. The expert system uses the knowledge base and behaviors facts to infer and give the results. This system can detect not only known malware, but some zero-day attacks using known techniques and also malware adopting low-level techniques, such as polymorphic and packer.

1. Introduction

Malware is a program with malicious intent that has the potential to harm the machine on which it executes or the network over which it communicates [1].The damage caused by malware has dramatically increased in the past few years [2]. So malware detection is a crucial aspect of software security. Nwokedi and Aditya [3] categorize techniques used for detecting malware broadly into two categories: anomaly-based detection and signature-based detection.

An anomaly-based detection technique uses its knowledge of what constitutes normal behavior to decide the maliciousness of a program under inspection. There are many anomaly-based detection techniques, including data mining approach [4,5], static analysis based approach [6,7], Finite State Automata (FSA) based approach[8], the frequency of system calls based approach [9], Fileprint (n-gram) analysis based approach [10], "cross-view diff-based" approach [11], Instruction Block Signatures based approach[12],etc. The key advantage of anomaly-based detection is its ability to detect zero-day attacks. But there are two fundamental limitations of this technique that are its high false alarm rate and the complexity involved in determining what features should be learned in the training phase.

Signature-based detection uses its characterization of what is known to be malicious to decide the maliciousness of a program under inspection. There are many signature-based detection techniques, including string signatures scanning [13], sequence of Windows API calls signatures based approach [14], templates signatures based approach [15], black-box signature based approach [16], etc. Currently, those signature-based detection techniques primarily rely on human expertise in creating the signatures that represent the malicious behavior exhibited by programs. Once a signature has been created, it is added to the signature-based method's knowledge. But the major drawback of the signature-based method for malware detection is that it cannot detect zero-day attacks.

At present, expert systems [17] used in information security are mainly in intrusion detection [18, 19, 20]. The mainly intrusion detection techniques are: misuse detection, anomaly detection and expert system based detection. Misuse detection has low false alarm because of its nature. But the main shortcomings of

Authorized licensed use limited to: Old Dominion University. Downloaded on October 12, 2009 at 10:40 from IEEE Xplore. Restrictions apply.

misuse detection are: known intrusion patterns have to be hand-coded; it is unable to detect any new or unknown attack that has no matched pattern stored in the system. Anomaly detection can detect new and unknown intrusion, but it has the shortcoming of false alarm rate. Expert system detection collects data through monitoring system, events, safety records and system records and intercepting the original IP packet. When the collected data showed suspicious activity, it will trigger the rules. The reasons for choosing an expert system are its accurate, high performance, real-time detection of the attacks, and it can give a detailed explanation of results and add new rules without changing existing rules and without creating any undesired dependency. Because of these reasons, we introduce expert system to our malware detection system.

In this paper, we propose a malware detection system based on expert systems. It uses expertise that is the suspicious behaviors of different kinds of malware. While detecting, the system uses its behaviors gathered component to gather behaviors happened in the host, and then use the CLIPS inference engine to reason. Our system integrates signature-based analysis and anomaly-detection technique. The signatures are suspicious behaviors, such as inline hook. So the advantages of our system are: its accurate and high performance; it can detect known or unknown malware no matter even it is hidden and resident; it also has considerable ability to detect malware adopting some circumvention techniques such as polymorphic, metamorphic, obfuscation and packer; and it can give a detailed explanation of results and add new rules without changing existing rules and without creating any undesired dependency.

The rest of the paper is organized as follows. In Section 2, profiles Expert Systems. In Section 3, presents our framework for malware detection. In Section 4 shows our experiment results and discussions. Section 5 concludes the paper.

2. Expert Systems

An expert system is a program which is specifically intended to model human expertise or knowledge. The expert system tool exploited in this paper is CLIPS (C Language Integrated Production System) [21]. CLIPS is an expert system shell originally developed in 1984 by the Artificial Intelligence Section of NASA's Johnson Space Center and is written in C. CLIPS uses a forward-chaining inference strategy based on the Rete pattern-matching algorithm. The CLIPS shell provides the basic elements of an expert system [22]:

1. *fact-list* and *instance-list*: Global memory for data
2. *knowledge-base*: Contains all the rules, the rule-base
3. *inference engine*: Controls overall execution of rules

A program written in CLIPS may consist of rules, facts, and objects. The inference engine decides which rules should be executed and when. A rule-based expert system written in CLIPS is a data-driven program where the facts, and objects if desired, are the data that stimulate execution via the inference engine.

Fact[17] is consists of relation name(a symbol field)、followed with zero or some slot(also symbol field) and their relevant value. The *deftemplate* [17] construct is used to create a template which can then be used by non-ordered facts to access fields of the fact by name.The syntax of the *deftemplate* construct [23] is:

```
(deftemplate <deftemplate-name> [<comment>]
        <slot-definition>*)
<slot-definition> ::=        <single-slot-definition>
        | <multislot-definition>
```

One of the primary methods of representing knowledge in CLIPS is an IF THEN rule. A *rule* [23] is composed of an *antecedent* and a *consequent*. The antecedent of a rule is a set of *conditions* (or *conditional elements*) which must be satisfied for the rule to be applicable. Rules are defined using the *defrule* construct [23].The syntax of the *defrule* construct is:

```
(defrule <rule-name> [<comment>][<declaration>] ;
        <conditional-element>* ; Left-Hand Side (LHS)
=>
        <action>*) ; Right-Hand Side (RHS)
```

3. Framework of Malware Detection System

Fig.1 shows the framework of Malware detection system based on expert system. The system consists of five main parts, *Knowledge Base*, *Fact List*, *Inference Engine*, *Behaviors Gathering Component*, *User Interface*. These parts are described in the following.

Authorized licensed use limited to: Old Dominion University. Downloaded on October 12, 2009 at 10:40 from IEEE Xplore. Restrictions apply.

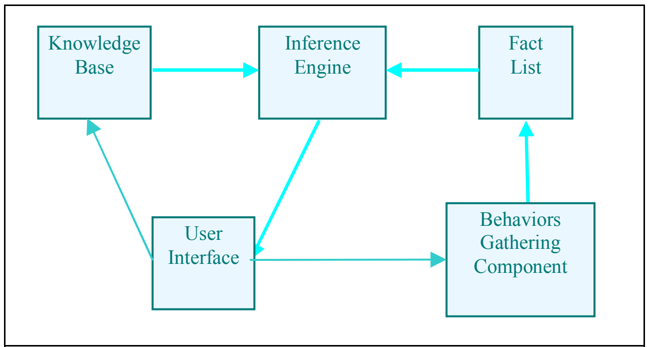

Fig.1. Framework of Our Malware Detection System

3.1. Knowledge Base

This part consists of domain knowledge and behavioral knowledge in the form of deductive rules. The domain knowledge is knowledge about the Operating System (Windows), including system processes list, system services list, file system, Windows Registry entries, etc. For example, the knowledge of system processes list:

```
(defclass SYS-PROC  (is-a USER)
                    (slot name))
(definstances system-process
(of SYS-PROC (name "C:\\WINDOWS\\system32\\csrss.exe"))
…
 (of SYS-PROC (name "C:\\WINDOWS\\system32\\smss.exe"))
)
```

The behavioral knowledge is expertise about what major suspicious behaviors of different kinds of malware. Using memory-resident malware as example, we discuss the construction of behavioral knowledge base. Such malware typically follow these steps:

1. The malware gets control of the system.
2. It allocates a block of memory for its own code.
3. It relocates its code to the allocated block of memory.
4. It activates itself in the allocated memory block.
5. It hooks the execution of the code flow to itself.
6. It infects new files and/or system areas.

This is the most typical pattern, but several other methods exist that do not require all of the preceding steps. Then the major suspicious behaviors of this kind of malware are: allocates a block of memory in some process's memory space; writes data to the allocated block of memory; does some kind of hook. Presents the behavioral signature knowledge as rules:

```
(defrule check-malicious-process
? allocate -memory <- (allocate-memory
          (srcPrc ?srcprc)(dstPrc ?dstprc&~?srcprc))
? write -memory <- (write-memory
          (srcPrc ?srcprc)(dstPrc ?dstprc&~?srcprc))
?inline-hook <- (inline-hook
          (scrAddr ?srcAddr)(dstAddr ?dstAddr)
          (dstName ?dstName)(modName ?modName))
(test(=(length$ (find-all-instances((?p SYS-PROCESS))
                    (eq ?p:name ?srcprc))) 0))
(not (malicious-process (process ?srcprc)))
=>
(printout t ?srcprc " is malicious..." crlf)
(assert (malicious-process (process ?srcprc))))
```

The templates using in the rules will be described in the part of *Fact List*.

3.2. Behaviors Gathering Component

The Behaviors Gathering Component is responsible for collecting various kinds of information data of the system, including behavioral information of malware using some low-level stealth techniques.

It consists of system services intercepted and trace scan. The system services intercepted can intercept efficiently amount of behaviors of malware in user mode or kernel mode. In order to achieve their aims, malware have some special behaviors, and these behaviors would leave a variety of significant traces. Scanning these traces can be effective with found and judge the presence of malware. But the intercepted behavioral data can not be directly delivered expert system modules. The data should be transformed to be presented as facts or instances which can be identified by the expert system. The examples of intercepted behavioral data are giving:

```
Create process:
          C:\hxdef100r\hxdef100.exe
          C:\WINDOWS\system32\smss.exe
Allocate memory:10000
          C:\hxdef100r\hxdef100.exe
          C:\WINDOWS\system32\smss.exe
          IsParent:1
Write memory:10000
          C:\hxdef100r\hxdef100.exe
   C:\WINDOWS\system32\smss.exe
```

The first one means process named hxdef100.exe whose path is "*C:\hxdef100r\hxdef100.exe*" creates a process named smss.exe whose path is "*C:\WINDOWS\system32\smss.exe*". The second means process hxdef100.exe allocates a block of memory whose base address is 10000 in the memory space of process smss.exe. The last one means process hxdef100.exe writes data to the memory space of process smss.exe, and the base address is 10000. The above examples after being preprocessed by this component will be:

```
(create-process
          (srcPrc "C:\WINDOWS\system32\services.exe")
          (dstPrc "C:\hxdef100r\hxdef100.exe"))
(allocate-memory
          (baseaddr 10000)
```

Authorized licensed use limited to: Old Dominion University. Downloaded on October 12, 2009 at 10:40 from IEEE Xplore. Restrictions apply.

```
            (srcPrc "C:\hxdef100r\hxdef100.exe")
            (dstPrc "C:\WINDOWS\system32\smss.exe"))
(write-memory
            (baseaddr 10000)
            (srcPrc "C:\hxdef100r\hxdef100.exe")
            (dstPrc "C:\WINDOWS\system32\smss.exe"))
```

The behavioral data after being preprocessed will be stored in the Fact List, for the Inference Engine module reasoning, to determine whether the procedures are malicious.

3.3. Fact List

The behavioral data after being preprocessed will be presented as facts. So we need define templates to describe facts. And the structure of each template is based on the behavioral signatures. For example, the "allocate memory" behavior, its description includes the following slots: the source process, the source process's PID value, the destination process, the destination process's PID value, and the base address being allocated. That is:

```
(deftemplate allocate-memory
            (slot baseaddress)(slot srcPid)
            (slot srcPrc)(slot dstPid)(slot dstPrc))
```

After we define these templates, the behavioral data gathered by Behaviors Gathering Component will be presented as facts using these templates.

3.4. Inference Engine

Inference Engine is a mechanism provided by CLIPS which automatically matches patterns against the current state of the fact-list and list of instances and determines which rules are applicable. Once a knowledge base (in the form of rules) is built and the fact-list and instance-list is prepared, CLIPS is ready to execute rules. In a conventional language, the starting point, the stopping point, and the sequence of operations are defined explicitly by the programmer. With CLIPS, the program flow does not need to be defined quite so explicitly. The knowledge (rules) and the data (facts and instances) are separated, and the inference engine provided by CLIPS is used to apply the knowledge to the data. The basic execution cycle [23] is as follows:

1. If the rule firing limit has been reached or there is no current focus, then execution is halted. Otherwise, the top rule on the agenda of the module which is the current focus is selected for execution. If there are no rules on that agenda, then the current focus is removed from the focus stack and the current focus becomes the next module on the focus stack. If the focus stack is empty, then execution is halted, otherwise step 1 is executed again.

2. The right-hand side (RHS) actions of the selected rule are executed. The use of the return function on the RHS of a rule may remove the current focus from the focus stack. The number of rules fired is incremented for use with the rule firing limit.

3. As a result of step 2, rules may be activated or deactivated. Activated rules (those rules whose conditions are currently satisfied) are placed on the agenda of the module in which they are defined. The placement on the agenda is determined by the salience of the rule and the current conflict resolution strategy. Deactivated rules are removed from the agenda. If the activations item is being watched, then an informational message will be displayed each time a rule is activated or deactivated.

4. If dynamic salience is being used, the salience values for all rules on the agenda are reevaluated. Repeat the cycle beginning with step 1.

3.5. User Interface

User interface is for users to interact with the expert system. It includes menu for the choice of knowledge base, button for starting reasoning and detecting, and the show column of detection results.

4. Experiments and discussions

This section details the experiments undertaken to evaluate the detection performance of our malware detection system. Since there are no other existing publicly available detection systems that based on expert systems we were unable to compare our performance to other system based on expert systems. To validate the claim that our detection technique is accurate, we compare our performance to some known anti-virus tools, such as Bitdefender, F-PROT, ESET NOD32, Kaspersky, McAfee and Norton.We choose Hxdef (Hacker defender) Rootkit [24] as malware sample. It is a user-mode rootkit that modifies several Windows and Native API functions, which allows it to hide information from other applications. We pack or protect Hxdef to generate some variants of Hacker defender. The packers used are: UPX, PeCompact, Upack, Petite, NsPack, FSG; and encryption protectors are: ACProtect ,ASProtect, Armadillo Custom, EXECryptor, MSLRH, Obsidium, PC Guard, PE-Armor, PeLock, PESpin, SDProtector, Themida, Winlicense, tElock, Yoda's Protector.

In our experiments, we use aforementioned packers and encryption protectors to process Hxdef, so we get 21 variants of Hxdef. Then we use 5 well known anti-

Authorized licensed use limited to: Old Dominion University. Downloaded on October 12, 2009 at 10:40 from IEEE Xplore. Restrictions apply.

virus tools, which are Bitdefender, F-PROT, ESET NOD32, Kaspersky and Norton, to scan these 22 malware samples. The scan results of these anti-virus tools are presented in Table.1. And then we run these malware samples and use our system presented in this paper to detect the system. The detection results are also showed in Table.1.

Object Name	Bitdefender	F-Prot	Kaspersky	Norton	Nod32	Our system
Hxdef	D	D	D	D	D	D
Hxdef _ACProtect	D	N	N	N	D	D
Hxdef _Armadillo	D	D	N	D	D	D
Hxdef _ASProtect	D	D	N	N	D	D
Hxdef _EXECryptor	N	N	N	N	N	D
Hxdef _FSG	D	D	D	D	D	D
Hxdef _MSLRH	N	D	D	N	D	D
Hxdef _Nspack	D	D	D	D	D	D
Hxdef _Obsidium	D	D	D	D	N	D
Hxdef _PCGuard	N	N	D	N	D	D
Hxdef _PEArmor	D^*	N	N	N	D	D
Hxdef _PEcompact	D	N	D	D	D	D
Hxdef _PeLock	D	D	D	N	N	D
Hxdef _PESpin	D^*	D	D	N	D	D
Hxdef _Petite	D	N	D	D	D	D
Hxdef _SDProtector	N	N	D	D	D	D
Hxdef _tElock	D	D	D	D	D	D
Hxdef _Themida	N	N	N	N	D^*	D
Hxdef _Upack	D	D	D	D	D	D
Hxdef _UPX	D	D	D	D	D	D
Hxdef _WinLicense	N	N	N	N	D^*	D
Hxdef _yoda Protector	D	D	D	D	D	D
Scanned files	22	22	22	22	22	22
Infected objects	16	13	15	12	19	22
Detect ratio	72.7%	59.1%	68.2%	54.5%	86.4%	100%

Table.1.Scan results of anti-virus tools and our system.

Note1: The "D" means the program is detected as malware; the "N" means the program is not detected, that is the anti-virus tool doesn't think it is suspicious.

Note2: The "D^*" means the tool just detects the program been packed by some packer, but can't judge whether the program is a threat or not.

From Table.1, we can see that our system's detection ratio is up to 100 percent. The detection ratio of the remaining anti-virus tools from high to low are Nod32 86.4%, Bitdefender 72.7%, Kaspersky 68.2%, F-PROT 59.1%, and Norton 54.5%. We need to pay attention to the "D^*" in Table.1, which means the tool just detects the program been packed by some packer or protector, but can't judge whether the program is a threat or not. In fact, many of the normal software use packer to process themselves in order to prevent reverse engineering and disassemble. Then if not including the "D^*", the detection ratio of Bitdefender and Nod32 will be 63.6% and 77.3%.

From the experiments, we can draw that our system can detect malware using known techniques, even low-level techniques, for example rootkit. It also can detect those malware even after they have being packed or encryption protected by any packer or protector. But the nature of the technique used in our system decides the main drawback of the system, which is the system can detect it just when the malware is running, otherwise it can't detect the malware. But these anti-virus tools do not require the malware is active.

5. Conclusions

This research has demonstrated suspicious behaviors can be valuable signatures in unknown malicious detection. Even if there are different malicious programs, such as Backdoor, but they all have similar behaviors. Using the system presented in this paper, we can detect malware using low-level techniques, such as rootkit, and these malware being packed by any packers or protectors. But the drawback of this method is we can detect malware just when it is running. So in order to having better performance, should use some static detection methods.

In future we intend to expand our behaviors knowledge base for more processes, and categorize behaviors signatures, so we can define and give the species of malware.

6. Acknowledge

This work is supported by the National 863 High-tech Program of China (No. 2006AA01Z449) and the 42nd National Science Foundation for Post-doctoral Scientists of China (No. 20070420738).

7. References

[1] M. D. Preda, M. Christodorescu, S. Jha, S. Debray. A Semantics-Based Approach to Malware Detection. POPL'07 January 17–19, 2007, Nice, France.

[2] R. Richardson, S. Peters. Computer Crime and Security Survey. Technical report, Computer Security Institute (CSI), 2007

[3] N. Idika, and A. P. Mathur, Survey of Malware Detection Techniques. Department of Computer Science Purdue University, February 2, 2007.

Authorized licensed use limited to: Old Dominion University. Downloaded on October 12, 2009 at 10:40 from IEEE Xplore. Restrictions apply.

[4] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In Proceedingsof the 7th USENIX Security Symposium, 1998

[5] Guangzhi Qu, Salim Hariri, Mazin S. Yousif. Multivariate statistical analysis for network attacks detection. AICCSA 2005: 9

[6] J. Bergeron, M. Debbabi, M.M. Erhioui, and B. Ktari. Static analysis of binary code to isolate malicious behavior. In 8th Workshop on Enabling Technologies: Infrastructure for Collaborative Entrerprises, 1999

[7] J. Bergeron, M. Debbabi, J. Desharnais, M.M. Erhioui, and N. Tawbi. Static detection of malicious code in executable programs. Int. J. of Req. Eng., 2001

[8] R. Sekar, M. Bendre, P. Bollineni, and D. Dhurjati. A fast automaton-based approach for detecting anomalous program behaviors. In IEEE Symposium on Security and Privacy, 2001

[9] I. Sato, Y. Okazaki, and S. Goto. An improved intrusion detection method based on process profiling. IPSJ Journal, 43:3316 – 3326, 2002

[10] W. Li, K.Wang, S. Stolfo, and B. Herzog. Fileprints: Identifying file types by n-gram analysis. 6th IEEE Information Assurance Workshop, June 2005

[11] Y. M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski. Detecting stealth software with strider ghostbuster. In Proceedings of the 2005 International Conference on Dependable Systems and Networks, pages 368–377, 2005

[12] M. Milenkovic, A. Milenkovic, and E. Jovanov. Using instruction block signatures to counter code injection attacks. ACM SIGARCH Computer Architecture News,33:108–117, March 2005

[13] Peter Szor .The Art of Computer Virus Research and Defence. Addison Wesley Professional,ISBN 0-321-30454-3,February 2005, Chapter 11

[14] A. Sung, J. Xu, P. Chavez, and S. Mukkamala. Static analyzer of vicious executables(save). In Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04), 00:326–334, 2004

[15] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, pages 32–46, 2005

[16] E. Filiol. Malware pattern scanning schemes secure against black-box analysis. Journal of Computer Virol., 2006

[17] Joseph C. Giarratano, Gary D. Riley. Expert Systems Principles and Programming Fourth Edition, February 22, 2006

[18] T.F. Lunt, A. Tamaru, F. Gilham et al. A real-time intrusion detection expert system (IDES) final technical report. Technical report, Computer Science Laboratory, SRI International, Menlo park, California, 1992.

[19] Debra Anderson, Thane Frivold, Alfonso Valdes. Next-generation Intrusion Detection Expert System (NIDES) A Summary. Computer Science Laboratory, SRI-CSL-95-07, 1995.

[20] De-gang Yang，Chun-yan Hu，Yong-hong Chen. A framework of Cooperating Intrusion Detection based on Clustering Analysis and Expert System. Proceeding of the InfoSecu04, Nov. 14-16, 2004.

[21] http://clipsrules.sourceforge.net/WhatIsCLIPS.html

[22] Joseph C. Giarratano, Ph.D. CLIPS User's Guide Version Quicksilver Beta,December 31st 2007. http://clipsrules.sourceforge.net/OnlineDocs.html

[23] CLIPS Reference Manua Volume I Basic Programming Guide Quicksilver Beta, December 31st 2007.http://clipsrules.sourceforge.net/OnlineDocs.html

[24] http://rootkit.host.sk/

Authorized licensed use limited to: Old Dominion University. Downloaded on October 12, 2009 at 10:40 from IEEE Xplore. Restrictions apply.