International Journal of Advanced Science and Technology
Vol.94 (2016), pp.1-12
http://dx.doi.org/10.14257/ijast.2016.94.01

Library Formation of Known Malicious Attacks and their Future Variants

Ajit Kumar Keshri[1], Bimal Kumar Mishra[1*] and Dheeresh K. Mallick[1]

Birla Institute of Technology, Mesra, Ranchi, India
ajitkeshri@bitmesra.ac.in, drbimalmishra@gmail.com, dkmallick@gmail.com

Abstract

In recent times, malicious objects have significantly increased in volume by many folds. Firewall, anti-virus (AV) or signature based intrusion detection systems (IDSs) all are found effective but only for known malicious objects. In that sense, unknown malicious objects are more dangerous. There are many techniques like honeypots, honeynets or anomaly based IDSs which are capable enough to identify these new malicious objects. This paper introduces a technique to develop a library not only for past and present attacks but also for future attacks, so that signature based IDSs not only detect known malicious objects but the unknown and future malicious objects as well. In our approach, with the help of a series of four algorithms, we show a way to develop all possible variants of each detected malicious object and finally update the library with these variants in order to empower it with future attacks.

Keywords: *Malicious Object, Computer Network, Data Mining, Data Compression*

1. Introduction

Malicious objects are referred by numerous names like malicious agents, malicious programs, malwares (abbreviation for malicious software) *etc*. They are harmful programs and when they infiltrate and infect any individual system or a network of systems using a variety of methods, they do damages like: reduce system performance or cripple the system components; destroy databases; stop or delete security programs; clean up log files to remove evidence of their presence; steal and send private information of the system to hackers; store or distribute illegal materials across the network; cause major disruption by increasing network traffic or clog the network *etc*.

According to their goal and properties such as self-propagating or non-propagating and self-replicating or non-replicating code, malicious objects can be classified into five major types as virus, worm, Trojan horse, spyware and bot and their subtypes. A virus is a self-reproducing automaton (SRA) that implants a version of itself to a normal program or file and then spreads to other files with user interaction. In contrast, a worm is an automated program that spreads copies of itself to compromised victims. But probably the main difference between them is their propagation model, that is, virus spread on a single system whereas worm spread via network connections makes it more dangerous as it can infect as many systems as possible [1]. A Trojan horse is a non replicating program and it entices to a useful and legitimate program, but containing hidden code meant to harm the system on which it is executed. A spyware is another non replicating malware which hide its presence to carry out malicious functions on a compromised victim. Spyware collects personal information and transmits the data across the network. A bot or zombie is the non replicating malicious object that operates automatically as agent

* Corresponding Author

ISSN: 2005-4238 IJAST

for a user. Perpetrators commonly known as hackers, forward bots to victims by a number of channels. The bots then wait for commands from a hacker, who can manipulate them and infect the system without the user's knowledge. A botnet is a network of compromised systems under the influence of bot, to provide illegal activities such as distributed denial of service (DDOS) attack [2].

According to their level of hiding capabilities, malicious objects can be further classified as i) Simple malicious objects ii) Encrypted malicious objects iii) Polymorphic malicious objects and iv) Metamorphic malicious objects. In Figure 1, they are arranged according to their level of hiding capabilities from low to high.

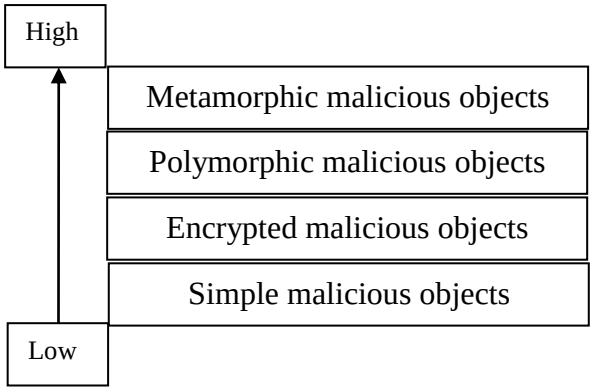

Figure 1. Level of Hiding Capabilities of Various Types of Malicious Objects

Simple malicious objects do not change their signature from infection to infection. Therefore, once a library is updated with their signatures, they can be detected very easily. In the realm of malicious object detection, signatures can be represented in binary sequences, octal sequences, n-gram (hexadecimal sequences), instruction sequences or application program interface (API) call (system call) sequences. To obtain these sequences except binary, intermediate representation of executable code is required. It can be achieved through various reverse engineering methods like disassembling, decompiling *etc* [3, 4]. These signatures can be created in many ways. Figure 2 shows an insight about signature creation. Here, by analyzing the input traffic, anomaly based IDS

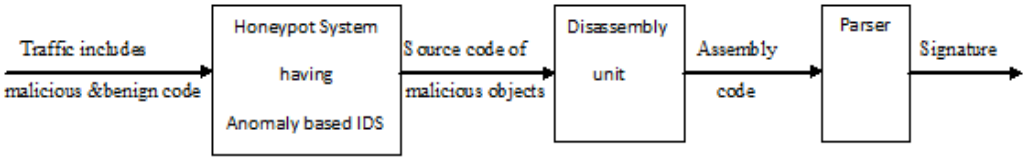

Figure 2. Steps to Generate One Type of Signature for Malicious Objects

try to detect malicious object. Like signature based IDS, anomaly based IDS consists of four parts as sensors, analysis center, databases and response center [5]. The databases of anomaly based IDS consists a white-list of the activities that represents the normal user behavior. In case, any deviation found in input traffic, its source code is handed over to the disassembly unit to convert this binary code into assembly code. Finally, a parser translates it into instruction sequence which only contains the opcode and the operands are discarded [6]. This instruction sequence is treated as a signature for that specific malicious object. Therefore, with the help of a signature library, AV or signature based IDSs can very easily detect simple malicious objects due to their fixed signature, whereas, encrypted malicious objects hide their fixed signature and using encrypted techniques make themselves

unrecognizable from being detected. They consist of two parts: decrypted routine and encrypted body. Whenever they infect a new file, they re-encrypt their body before and then append themselves to files which make them different in every infection. Though encrypted body changes from infection to infection, its decryption routine remains constant. Therefore, the same signature detection techniques which are used for simple malicious objects can then be used to identify decrypted routine in order to identify encrypted malicious objects. Polymorphic malicious objects have mutation engine along with decryption routine and encrypted body. This mutation engine generates a new decryption routine whenever they infect a new file. With no constant decryption routine, it's almost impossible to detect them. Several antivirus companies have incorporated complex technique called generic decryption (GD) into their virus scanner to detect polymorphic malicious objects [7, 8]. Metamorphic malicious objects change their code structure from infection to infection by using all or any of the three techniques as inserting junk code, replacing instructions with equivalent operations and changing the order of code blocks [9]. Hiding techniques like variable renaming and change of conditional jumps are also used by metamorphic malicious objects to evade detection [10]. Therefore, it is next to impossible for signature based IDSs to detect metamorphic malicious objects. Although the signature of metamorphic malicious objects changes due to the above mentioned techniques, their behavior remains unchanged. Behavior based IDSs, based on characteristics set that describe the behavior, can be used to identify this type of malicious objects.

Though many detection techniques including above mentioned techniques are in existence, they have some short of disadvantages. These defense mechanisms are especially inadequate to defend unknown malicious objects. Therefore, the ever increasing volume of malicious objects, largely due to the variants of known malicious objects, remains the greatest challenge for anti- virus researchers.

Malicious objects can infect any system by calling critical API calls. Critical API calls are those API calls that can lead to security compromise. Based on these critical API calls (like Registry API, Winsock API, File I/O API *etc*.,) and their frequency, the behavior of malicious objects are identified and classified in [11]. Another technique is to create and publish patches for exploited vulnerability, so that network systems can safeguard themselves by downloading and install these patches. Since applied patches require human reactions, it is not an effective technique. Castaneda *et al.*, [12] has proposed an immunization method by distributing patches with the help of anti-worms. This method transforms a malicious worm into a good worm called anti-worm which uses the same vulnerability to infect other systems and then download the appropriate patch to immunize them. But due to legal issues this counter-attack mechanism is not accepted as a defense mechanism. In any attack, malicious and benign signals are mixed, so the resulting signals are weak and hard to detect. Ye and Farlay [13] suggested that the way signal detection models separate signals from a mix of signal and noise, using the same models, malicious signals can be separated from benign signals. However, the subject is far from mature and key issues remains to be solved.

The remainder of this paper is structured as follows. Section 2 presents the architecture of our approach. Section 3 elaborates each of our four proposed algorithms. It gives inside about signature creation, signature compression and variant generation. In Section 4, we have tested our algorithms with the help of a real boot sector malicious object. Finally, Section 5 provides concluding remarks.

2. Proposed Architecture

How resistant is a signature based IDS to variants of known malicious objects? Unfortunately, current signature based IDSs are only capable of detecting known malicious objects because unlike anomaly based IDSs, there databases consist a black-list of known activities, only whom they can detect. Our proposed approach, as shown in the Figure 3, will help signature based IDSs to detect not only known malicious objects but their unseen variants as well. In stage 1 of our architecture, we have used honeypot as a decoy to trap and detect malicious objects. Honeypot is an unprotected system having unused IP address which get compromised very easily and since it is an unadvertised system with no services offered, all traffic on a honeypot can be considered as malicious. It is not only used to decoy malicious attacks for detection of both types of malicious objects (known and unknown) but also to log and analyze inbound and outbound traffic data to learn about attackers, their malicious activities, methodologies and motivations.

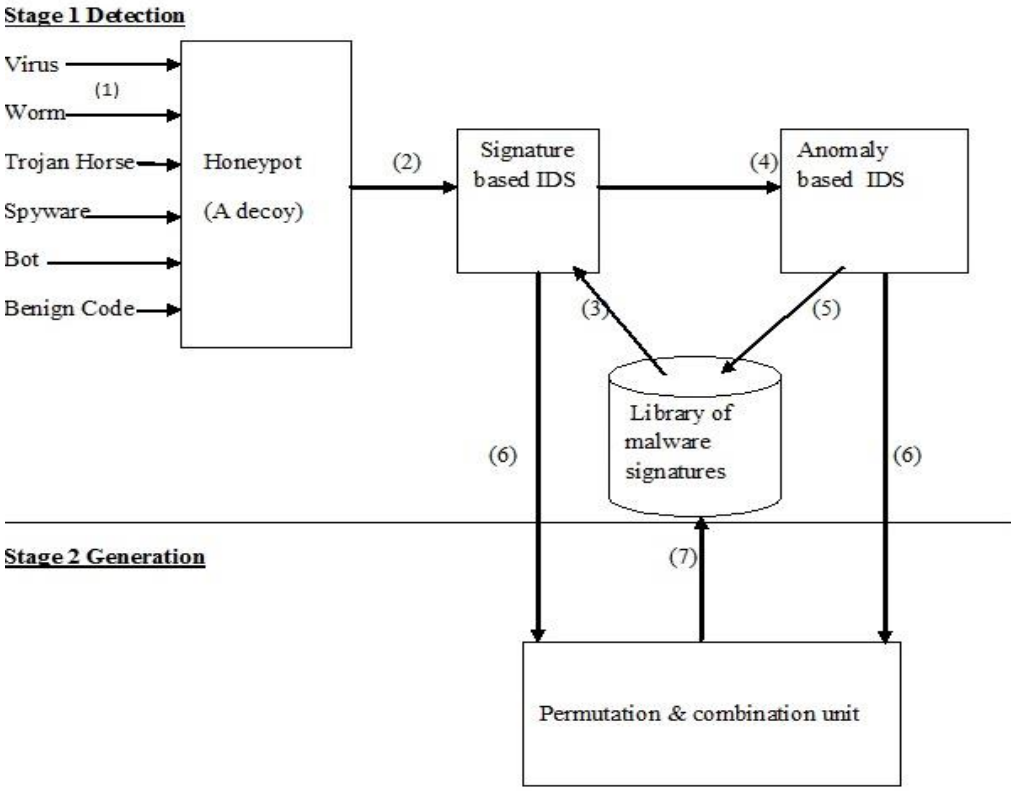

Figure 3. Architecture for Library of Malicious Objects Signatures

Use of cunning mechanisms like deception [14, 15] and camouflage [16] in honeypot can improve its success to achieve its goals. Honeypots can be designed using deception so that they seem like a real system. The trick is to keep the attackers fooled as long as possible so that more and more malicious codes can be collected in order to empower our library with present attacks. At step (1), a message enters the honeypot system but it is not known if it is malicious or not. In step (2), the message is handed over to the signature based IDS to test if it is known malicious object or not by comparing its signature (step (3)) with the library of malware signatures. If the signature does not match, then in step (4), the same message is handed over to the anomaly based IDS. It then analyzes the message, based on its behavior or characteristic sets, and finally if the message is found as a

malicious code, in step (5) the signature of this malicious object is updated in the library. At step (6), anomaly based IDS send the signature to the permutation unit to generate all possible variants of detected malicious object and at step (7), all these variants are updated in the library of malicious signatures.

3. Proposed Algorithms

Here, we have proposed four algorithms to achieve our goal. First algorithm provides the steps for signature creation of the detected malicious object, whereas, second algorithm gives a data compression technique to compress the signature derived in algorithm one. From this compressed signature, some instructions which are common in benign codes and malicious object codes are removed. Therefore, signature size may get reduced further. Third algorithm is designed to achieve this. Finally, our fourth algorithm provides the steps to create all possible variants of it. The variant generation module is shown in Figure 4.

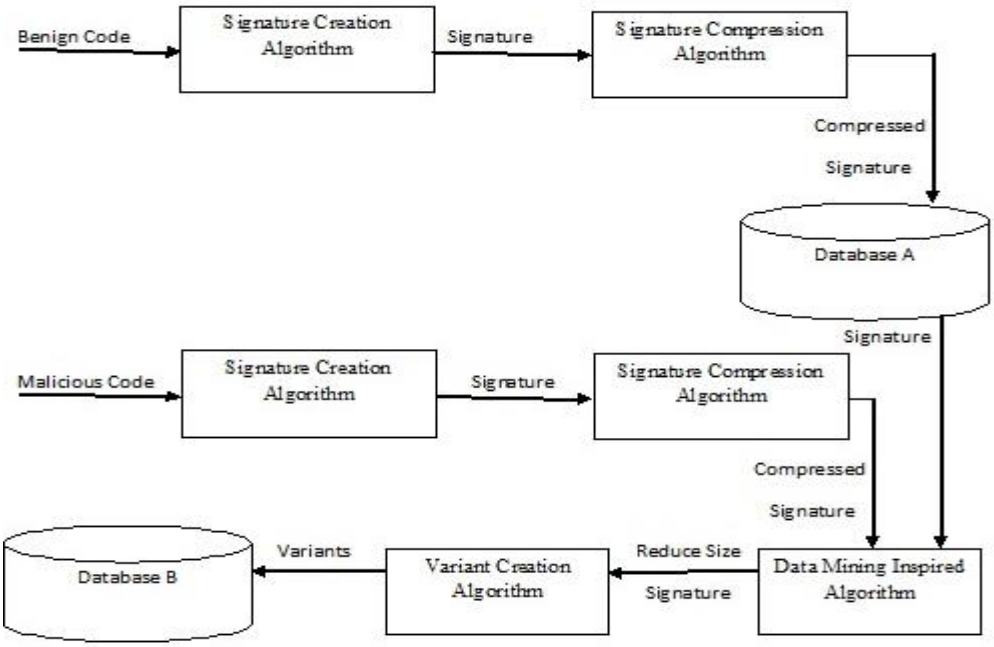

Figure 4. Variant Generation Module

3.1. Signature Creation Algorithm

The steps of signature creation algorithm are as follows:

Step 1: Read the source code.

Step 2: Convert it to assembly code with the help of a disassembly unit.

Step 3: With the help of a parser only keep the opcode and delete the operands and other parts (like addressing mode etc.) from each instruction.

Step 4: Read the opcode sequentially from top to bottom and arrange them horizontally such that they are separated from each other by a space.

Step 5: Move from left to right to check any opcode is equal to any of the following: B, BX (where X can be EQ, NE, LT, LE, GT, GE, LO LS, HI, HS, VS, VC, MI, PL, L), CALL, RET, INT, LOOP, JNZ, JZ, JMP. If so, break the line after

that opcode and place the rest in the next line. Continue the process till all the opcodes are checked. The final outcome of this step is called signature.

3.2. Signature Compression Algorithm

Total number of distinct opcodes in any computer architecture is very small in numbers. So, we have used hypothetical sequence of hashes to represent them. We have used 8 bits to assign distinct hash to each opcode i.e., 2^8 or 256 distinct hashes are available which is more than enough for all types of opcode found in ARM architecture, MIPS architecture, x86 architecture or any other architecture. This technique helps to reduce the signature size. Therefore, in this signature compression algorithm, we reduce the size of any signature, created by proposed signature creation algorithm, by replacing each opcode with their corresponding distinct hash. The steps of this algorithm are as follows:

Step 1: Read the signature from signature creation algorithm.

Step 2: Replace each and every opcode by their respective distinct hash. Now, the outcome is a compressed signature.

3.3. Data Mining Inspired Algorithm

A bunch of benign programs, which can include original operating system (OS) files and other clean applications, need to be considered and using the above mentioned two algorithms (in sections 3.1 and 3.2), their signature need to be generated and stored in a database called A and the same is shown in first part of Figure 4. This is pre-requisite for data mining inspired algorithm.

Now in this algorithm, we take the malicious object's compressed signature which is generated by second algorithm. Each instruction or line of this signature is compared with the instructions of database A and the common instructions are dropped from the signature. The remaining instructions construct the final signature of that malicious object. Therefore, after Section 3.2, also in this section the signature size of malicious objects reduce significantly. The steps of this algorithm are as follows:

Step 1: Read the signature from the signature compression algorithm.

Step 2: Compare each instruction of the signature with the signatures stored in database A and remove the common instructions from the signature. Now, the outcome is the final signature of that malicious object.

3.4. Variant Creation Algorithm

In this section, we will find all possible distinct arrangements first among the numbers of each instruction and then among all instructions. The number of all possible variants can be calculated by multiplying all the calculated arrangements. The steps of this algorithm are as follows:

Step 1: Read the final signature from the third algorithm.

Step 2: Read an instruction from the signature. If it contains ***n*** numbers, then only ***(n-1)*** numbers from left are considered for permutation. It may have two cases: i) all the ***(n-1)*** numbers are distinct or ii) there is repetition of some numbers. For first case, go for ${}^{(n-1)}P_{(n-1)}$ to get all possible different arrangements among them. For second case, if there are ***(n-1)*** numbers, of which p_1 are alike of one kind; p_2 are alike of another kind; …p_r are alike of ***r***th kind such that $p_1 + p_2 + \ldots p_r = (n-1)$; then the number of permutations of these ***(n-1)*** numbers is $\frac{(n-1)!}{(p_1!)\times(p_2!)\times\ldots.\times(p_r!)}$.

Step 3: Repeat step 2 for the rest of instructions.

Step 4: Assume the signature contains ***m*** number of lines or instructions. Therefore, to calculate all possible arrangements of them, permutation is required. It also has two cases: i) all the instructions are distinct or ii) there is repetition of some instructions. Like step 2, for first case, go for ${}^{m}P_{m}$ to get all possible different arrangements among them and for second case, if there are ***m*** instructions, of which p_1 are alike of one kind; p_2 are alike of another kind; …p_r are alike of ***r***th kind such that $p_1 + p_2 +_{\ldots} p_r = m$; then the number of permutations of these ***m*** instructions is

$$\frac{m!}{(p_1!)\times(p_2!)\times \ldots. \times (p_r!)}.$$

4. Experiments

Our experiment is divided into three parts. In part one, we have created a compressed signature for a boot sector virus using the first two algorithms. In the next part, we have developed the same compressed signature but this time it is for some benign codes and stored the signatures in the database A as shown in the Figure 4. In the last part, we have removed the common instructions from the virus signature by comparing it with the signatures available in the database A. The remaining code is the final signature of the boot sector virus. Finally, using our last algorithm we have created all possible variants of this signature.

4.1. Part One: Compressed Signature of a Boot Sector Virus

To test our algorithms, we have selected a simple boot sector virus named Kilroy virus [17]. Boot sector is the first code to gain control after the read only memory (ROM) startup code. A boot sector virus with sophisticated anti-detection routines can be very difficult to detect and makes it nearly invincible. If any disk gets infected by this Kilroy virus, a message consisting of "Kilroy was here" is displayed.

Ludwig's HEX (hexadecimal) listing of the Kilroy virus is as follows:

```
:10000000EB28904B494C524F59202000020201002E
:10001000027000D002FD020009000200000000092
:1000200000001200000000010000FA33C08EC08EF4
:10003000D0BC007CBB780036C5371E561653BF1E99
:100040007CB90B00FCAC26803D007503AAEB014790
:10005000E2F38AC48ED8894702C7071E7CFBCD1302
:1000600072FEE83E01BB0005803EFD7D80742EBA25
:100070008001803E7504007424B90100B80102CDEE
:1000800013721A813EFE0655AA7512E8FE00BA8068
:1000900001B90100B80103CD137202EB32A01004C4
```

Keeping the ethics in mind, only a small part of the HEX code is considered for our academic purpose. Now, using our first algorithm, step by step we have created the signature of this Kilroy virus as follows:

Step 1: Read the binary version of the complete Kilroy virus code.

Step 2: Convert this source code to its equivalent assembly code. A portion of the generated assemble code is shown in Figure 5.

```
SPREAD:
CALL DISP_MSG
MOV BX,OFFSET DISK_BUF
CMP BYTE PTR [DRIVE],80H
JZ SPREAD2
MOV DX,180H
CMP BYTE PTR [HD_COUNT],0
JZ SPREAD2
MOV CX,1
MOV AX,201H
INT 13H
JC SPREAD2
CMP WORD PTR [NEW_ID],0AA55H
JNZ SPREAD2
CALL MOVE_DATA
MOV DX,180H
MOV CX,1
MOV AX,301H
INT 13H
JC SPREAD2
JMP SHORT LOOK_SYS
```

Figure 5. Assembly Code Generated at Step 2 of Signature Creation Algorithm

Step 3 and 4: Only opcodes are considered in step 3 and then in step 4 they are arranged horizontally, as shown in Figure 6.

```
CALL MOV CMP JZ MOV CMP JZ MOV MOV INT JC CMP JNZ CALL MOV MOV MOV
INT JC JMP
```

Figure 6. Result Obtained from Step 4 of Signature Creation Algorithm

Step 5: This single line code is divided into multiple lines as per the condition specified in the algorithm. The resulted signature after this final step for our example is shown in Figure 7.

Now, our second algorithm helps to compress this signature as follows:

Step 1 and 2: Each opcode of signature shown in Figure 7 are replaced by their corresponding distinct hash as stated in the algorithm. The result is shown in Figure 8 by assuming some distinct hypothetical hashes.

```
CALL
MOV CMP JZ
MOV CMP JZ
MOV MOV INT
JC
CMP JNZ
CALL
MOV MOV MOV INT
JC
JMP
```

Figure 7. Signature of Kilroy Virus Produced by Signature Creation Algorithm


```
24
11 17 26
11 17 26
11 11 28
27
17 29
24
11 11 11 28
27
25
```

Figure 8. Compressed Signature of Kilroy Virus Produced by Signature Compression Algorithm

4.2. Part Two: Compressed Signature of a Set of Benign Programs

Here, we have considered parts of two benign programs, the assembly codes of which are shown in Figure 9 and 10. Then we have applied both the algorithms to get their compressed signatures which are shown in Figure 11 and 12. Finally these signatures are stored in database A.

```
Main:
MOV R9, #5
MOV R0, R9
BL my_alloc
MOV R10,R0
MOV R1, R9
BL my_init
MOV R0, R10
MOV R1,R9
BL sort
```

Figure 9. Assembly Code of First Benign Program

```
For1tst:
CMP i, n
BGE exit1
SUB j, i, #1
For2tst:
CMP j, #0
BLT exit2
ADD vjAddr, v, j, LST #2
LDR vj, [vjAddr, #0]
LDR vj1, [vjAddr, #4]
CMP vj,vj1
BLE exit2
MOV r0, vcopy
MOV r1, j
BL swap
SUB j, j, #1
B for2tst
EXIT2:
ADD i, i, #1
B for1tst
```

Figure 10. Assembly Code of Second Benign Program

```
11 11 21
11 11 21
11 11 21
```

Figure 11. Compressed Signature of First Benign Program

```
17 18
1 17 18
0 2 2 17 18
11 11 21
1 19
0 19
```

Figure 12. Compressed Signature of Second Benign Program

4.2. Part Three: Creation of Final Signature and Variants Generation of Kilroy Virus

As per the data mining inspired algorithm, here the compressed signature of Kilroy virus gets compared with the signatures stored in database A. Since no match was found in our example, the final signature of Kilroy virus remains same as its compressed signature. The size would have reduced to a great extent if we have sufficient signatures available in database A. Now, from this final signature we have derived 3628800 variants by using our final algorithm. For example, two derived variants are shown in Figure 13 and 14.

```
24
17 11 26
11 17 26
27
27
17 29
11 11 28
24
11 11 11 28
25
```

Figure 13. Variant Number 1 of Kilroy Virus Signature Produced by Variant Creation Algorithm

```
11 11 28
27
25
17 11 26
17 29
24
11 11 11 28
17 11 26
27
24
```

Figure 14. Variant Number 2 of Kilroy Virus Signature Produced by Variant Creation Algorithm

5. Conclusion

In this paper, we have addressed a problem raised due to frequent outbreaks of malicious object variants. To encounter this problem, we proposed a method which is based on a multi-layer defense by putting a signature based IDS as the first layer defense and an anomaly based IDS as the second layer defense. This method is used to generate signatures for all known malicious objects and their all possible variants including unknown and future type and finally update these signatures to our library, so that previously unseen malicious objects along with known malicious objects can be detected.

References

[1] N. Idika and A. P. Mathur, "A Survey of Malware Detection Techniques", Technical report, Department of Computer Science, Purdue University **(2007)**, pp. 1-48.
[2] H. R. Zeidanloo, M. J. Z. Shooshtari, P. V. Amoli, M. Safari and M. Zamani, "A Taxonomy of Botnet Detection Techniques", Computer Science and Information Technology (ICCSIT), 3rd IEEE International Conference, vol. 2, **(2010)** July, pp. 158-162.
[3] M. Sharif, A. Lanzi, J. Giffin and W. Lee, "Automatic Reverse Engineering of Malware Emulators", Proceedings of the IEEE Symposium on Security and Privacy, **(2009)**, pp. 94-109.
[4] M. A. Siddiqui, "Data Mining Methods for Malware Detection", Ph.D. Thesis, University of Central Florida, **(2008)**.
[5] A. A. Cardenas, T. Roosta, G. Taban and S. Sastry, "Cyber Security Basic Ddefenses and Attack Trends", G. Franceschetti, M Grossi, Homel and Security Technology Challenges, Artech House Inc., Massachusetts, USA, **(2008)**, pp. 73-101.
[6] M. A. Siddiqui, M. C. Wang and J. Lee, "Detecting Internet Worms using Data Mining Techniques", Journal of Systemics Cybernetics and Informatics, vol. 6, no. 6, **(2008)**, pp. 48-53.
[7] C. Nachenberg, "Computer Virus-antivirus Coevolution", Communications of the ACM, vol. 40, no. 1, **(1997)**, pp. 47-51.
[8] C. Nachenberg, "Understanding and Managing Polymorphic Viruses", Technical Report, The Symantec Enterprise Papers, vol. 30, **(1996)**.
[9] F. Leder, B. Steinbock and P. Martini, "Classification and Detection of Metamorphic Malware using Value Set Analysis", Proceedings of 4th International Conference on Malicious and Unwanted Software: MALWARE, Montreal, Canada, **(2009)**.
[10] M. Christodorescu and S. Jha, "Testing Malware Detectors", Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Boston, Massachusetts, USA, **(2004)**, pp. 34-44.
[11] V. S. Sathyanarayan, P. Kohli and B. Bruhadeshwar, "Signature Generation and Detection of Malware Families", Proceedings of the 13th Australasian conference on Information Security and Privacy, LNCS 5107, Springer, **(2008)**, pp. 336-349.
[12] F. Castaneda, E. C. Sezer and J. Xu, "Worm vs. Worm: Preliminary Study of an Active Counter-attack Mechanism", Proceedings of the ACM workshop on rapid malcode, **(2004)**, pp. 83-93.
[13] N. Ye and T. Farley, "A Scientific Approach to Cyber Attack Detection", IEEE Computer Society, vol. 38, no. 11, **(2005)**, pp. 55-61.
[14] N. C. Rowe, "Designing Good Deceptions in Defense of Information Systems", Proceedings of the Annual Computer Security Applications Conference (ACSAC), Tucson, AZ, **(2004)**, pp. 418-427.
[15] F. Cohen, "The Use of Deception Techniques: Honeypots and Decoys", Bidgoli, H. (ed.) Handbook of Information Security, John Wiley & Sons, Chichester, vol. 3, **(2006)**, pp. 646-655.
[16] X. Fu, B. Graham, D. Cheng, R. Bettati and W. Zhao, "Comouflaging Virtual Honeypots", Texas A&M University, Technical Report, **(2005)**.
[17] M. A. Ludwig, "The Little Black Book of Computer Viruses", American Eagle, Arizona, **(1996)**.

Authors

Ajit Kumar Keshri received his post-graduate degree [M. Tech. (CS)] from Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India in 2002 and his under-graduate degree [B.E.(Civil Engineering)] from Bangalore University, Bangalore, Karnataka, India in 1999. He is presently serving as an Assistant Professor in the Department of Computer Science and Engineering at the Birla Institute of Technology, Mesra, Ranchi, Allahabad Campus (India). He has a total 12 years of academic experience. His area of interest for research includes mathematical modeling of cyber attacks and defense in wireless network.

Bimal Kumar Mishra is presently serving as a Professor in the Department of Mathematics at the Birla Institute of Technology, Mesra, Ranchi Jharkhand, India. After completing his doctoral degree in Mathematics in the year 1997, he subsequently obtained D. Sc. in Mathematics in the year 2007. He has been actively involved in both teaching and research for almost two decades. His area of interest for research includes mathematical modeling of cyber attacks and defense, non-linear dynamical systems and their stability, and also the study of infectious diseases. He has presented his work through more than hundred research publications in various journals of international repute and conferences.

Dheeresh K. Mallick received Ph. D. in Computer Science in 2010 from Indian School of Mines Dhanbad, India. He was awarded Master of Technology degree in Computer Science from Birla Institute of Technology, Mesra, Ranchi in 2000. He is an Associate Professor in the Department of Computer Science and Engineering, Birla Institute of Technology at Ranchi, India. His primary research interests include paralle lalgorithms, interconnection networks, optoelectronic parallel computing and WSN. He has contributed 19 research papers in International Journals and Conference Proceedings.

