ISSN : 2455-9679
Impact Factor : 3.340

IJRTSM

INTERNATIONAL JOURNAL OF RECENT TECHNOLOGY SCIENCE & MANAGEMENT

"A REVIEW ON CODE COMPLICATION AND VIRUS DETECTION"

Mayank Singh [1]*, Dr. Anand K. Tripathi* [2]

[1] *P. G. Student, Department of Computer Science Engineering, P K University, Shivpuri, Madhya Pradesh, India*
[2]*Assistant Professor, Department of Computer Science Engineering, P K University, Shivpuri, Madhya Pradesh, India*

ABSTRACT

Traditional viruses were computer programs with static structure exhibiting very limited functionality. Once identified for the first time, their structure is utilized by antivirus (AV) software as a tool for detecting the similar viruses with similar patterns. However, modern viruses are smart enough to self-configure and even change the pattern of their functionality making it hard for AV software detecting them. This paper shows that to develop new reliable antivirus software some problems must be solved such as: a new method to detect all metamorphic virus copies, new reliable monitoring techniques to discover the new viruses or attaching a digital signature and a certificate to each new software.

Key Words: *Antivirus, Digital Signature, Dynamic Detection, Metamorphic Virus and Static Detection.*

I. INTRODUCTION

In today's world, where a majority of the transactions involving sensitive information access over the internet, it is absolutely imperative to treat information security as a concern of importance. Computer viruses and other malware have been in existence from the very early days of the pc and continue to pose a threat to home and enterprise users. When Anti-Virus techniques came to remove or detect these viruses, the virus developer also changed their strategies to develop more complex and nearly impossible to detect viruses.
Both viruses and virus detectors have gone through several phases of change since the first appearance of viruses and this thesis is concerned with a recent stage in virus evolution—metamorphic viruses. These are viruses which employ code Complication techniques to hide and mutate their appearance in host programs as a means to avoid detection.Signature based static detection is the most famous virus detection technique employed today is, which involves looking for a fingerprint-like sequence of bits (extracted from a known sample of the virus) in the suspect file. Metamorphic viruses are quite potent against this technique since they can create variants of themselves by code-morphing and the morphed variants do not necessarily have a common signature. In fact, the paper [1] provides a rigorous proof that metamorphic viruses can bypass any signature-based detection, provided the code Complication has been done based on a set of specified rules.

II. STRATEGIES OF COMPUTER

A computer virus is a computer program that can copy itself and infect a computer without permission or knowledge of the user. In order to avoid detection by users, some viruses employ different kinds of deception such as the following Strategies.

ISSN : 2455-9679
Impact Factor : 3.340

• **Overwriting Virus**: this type of virus overwrites files with their own copy. Of course, this is a very primitive technique, but it is certainly the easiest approach of all. Overwriting viruses cannot be disinfected from a system. Infected files must be deleted from the disk.

• **Companion Infection:** one approach to becoming a companion to an EXE file is to give the virus the same base name as the targeted program, but use a .COM extension instead of .EXE. This technique was employed by the Globe virus, first detected in 1992. When the victim attempts to launch an EXE program, he or she usually types its name without the extension. In such cases, Windows gives priority to a file with the .COM extension over a file with the same base name but with the .EXE extension.

• **Appending Virus:** In this technique, a jump (JMP) instruction is inserted at the front of the host to point to the end of the original host. A typical example of this virus is Vienna. The appender technique can be implemented for any other type of executable file, such as EXE, NE, PE, and ELF formats, and so on. Such files have a header section that stores the address of the main entry point, which, in most cases, will be replaced with a new entry point to the start of the virus code appended to the end of the file.

• **Prepending Virus:** This virus inserts its code at the front of host programs. This is a simple kind of infection, and it is often very successful. writers have implemented it on various operating systems, causing major virus outbreaks in many. An example of a COM prepender virus is the Hungarian virus Polimer.512.A, which prepends itself, 512 bytes long, at the front of the executable and shifts the original program content to follow itself.

• **Cavity or space filler Virus:** This virus attempts to install itself in this empty space while not damaging the actual program itself. An advantage of this is that the virus then does not increase the length of the program and can avoid the need for some stealth techniques. The Lehigh virus was an early example of a cavity virus. Because of the difficulty of writing this type of virus and the limited number of possible hosts, cavity viruses are rare.

• **Compressing Virus:** A special virus infection technique uses the approach of compressing the content of the host program. Sometimes this technique is used to hide the host program's size increase after the infection by packing the host program sufficiently with a binary packing algorithm.

• **Encrypted Virus:** consists of a constant decryptor, followed by the encrypted virus body. Relatively easy to detect because decryptor is constant. The first known virus that implemented encryption was Cascade on DOS. Oligomorphic virus changes its decryptors in new generations. The simplest technique to change the decryptors is to use a set of decryptors instead of a single one. The first known virus to use this technique was Whale. Whale carried a few dozen different decryptors, and the virus picked one randomly.

• **Boot Sectors Virus:** this virus takes advantage of the executable nature of master boot record (MBR) and partition boot sector (PBS). A PC infected with a boot sector virus will execute the virus's code when the machine boots up. Michelangelo virus is an example of a Boot Sectors Virus.

• **macro virus:** infects a Microsoft Word or similar application and causes a sequence of actions to be performed automatically when the application is started or something else triggers it. Macro viruses tend to be surprising but relatively harmless. A typical effect is the undesired insertion of some comic text at certain points when writing a line. A macro virus is often spread as an e-mail virus. A well-known example in March, 1999 was the Melissa virus.

• **Malicious mobile code (MMC):** mobile code is a lightweight program that is downloaded from a remote system and executed locally with minimal or no user intervention. Java applets, JavaScript scripts, Visual Basic Scripts (VBScripts), and ActiveX controls are some of the most popular examples of mobile code that you may encounter while browsing the Web or reading HTML-formatted e-mail. An attacker might use mobile code for a variety of nasty activities, including monitoring your browsing activities, obtaining unauthorized access to your file system, infecting your machine with a Trojan horse, hijacking your Web browser to visit sites that you did not intend to visit, and so on.

III. LITERATURE REVIEW

P. Denning et al [1983] Trojan horse, computer viruses are instances of malicious logic or malicious programs. Other programs which may be malicious but are not computer viruses are worms, which copy themselves from computer to computer4 ; bacteria, which replicate until all available resources of the host computer are absorbed; and logic bombs, which are run when specific conditions, such as the date being Friday the 13th, hold. Malicious logic uses the user’s rights to perform their functions; a computer virus will spread only as the user’s rights will allow it, and can only take those actions that the user may take, since operating systems cannot distinguish between intentional and unintended actions.[1]

ISSN : 2455-9679
Impact Factor : 3.340

B. Randel et al [1978] Systems implementing multilevel security and integrity policies usually allow some small set of trusted entities to violate the stated policy when necessary for the smooth operation of the computer system. The usefulness of whatever security model the system implements depends to a very great extent on these exceptions; for should a trusted entity attempt to abuse its power to deviate from the strict policy, little can be done. The statements describing the effects of the controls on malicious logic above apply only to the model, and must be suitably modified for those situations in which a security policy allows (trusted) entities to violate the policy. The two phases of a computer virus' execution illustrate this. Infecting (altering) a program may be possible due to an allowed exception to the site's integrity model. Executing a computer virus to disclose some information across protection domain boundaries may also be possible because of an allowed exception to the site's disclosure model. So the virus may spread more widely because of the allowed exceptions. An alternate view of malicious logic is that it causes the altered program to deviate from its specification. If this is considered an "error" as well as a breach of security, fault-tolerant computer systems, which are designed to continue reliable operation when errors occur, could constrain malicious logic. Designers of reliable systems place emphasis on both recovery and preventing failures [2]

] K. Thompson et al [1984] Ken Thompson created a far more subtle replicating Trojan horse when he rigged a compiler to break login security. When the compiler compiled the login program, it would secretly insert instructions to cause the resulting executable program to accept a fixed, secret password as well as a user's real password. Also, when compiling the compiler, the Trojan horse would insert commands to modify the login command into the resulting executable compiler. Thompson then compiled the compiler, deleted the new source, and reinstalled the old source. Since it showed no traces of being doctored, anyone examining the source would conclude the compiler was safe. Fortunately, Thompson took some pains to ensure that it did not spread further, and it was finally deleted when someone copied another version of the executable compiler over the sabotaged one. Thompson's point was that "no amount of source-level verification or scrutiny will protect you from using untrusted code" , which bears remembering, especially given the reliance of many security techniques relying on humans certifying programs to be free of malicious logic. [3]

F. Cohen et al [1987] Fred Cohen designed a computer virus to acquire privileges on a VAX-11/750 running UNIX; he obtained all system rights within half an hour on the average, the longest time being an hour, and the least being under 5 minutes. Because the virus did not degrade response time noticeably, most users never knew the system was under attack. In 1984 an experiment involving a UNIVAC 1108 showed that viruses could spread throughout that system too. Viruses were also written for other systems (TOPS-205 , VAX/VMS, and a VM/3706 system) but testing their effectiveness was forbidden. Cohen's experiments indicated that the security mechanisms of those systems did little if anything to inhibit computer virus propagation [4].

T. Duff et al [1989] Tom Duff experimented on UNIX systems with a small virus that copied itself into executable files. The virus was not particularly virulent, but when Duff placed 48 infected pro grams on the most heavily used machine in the computing center, the virus spread to 46 different systems and infected 466 files, including at least one system program on each computer system, within eight days. Duff did not violate the security mechanisms in any way when he seeded the original 48 programs [5]

Lee and Mody [2006] propose a system that divides a body of malicious software samples into clusters by applying machine learning techniques on behavioral profiles of the samples. The execution of these samples take place in a tightly controlled virtual environment. A kernel-mode monitor records all system call invocations along with their arguments. The retrieved information about a sample's interaction with the system is recorded into a behavioral profile. This profile consists of information regarding the sample's interaction with system resources such as writing files, registry keys, or network activity. To measure the similarity between two profiles, the edit distance is calculated between them, where the cost of a transformation is defined in an operation cost matrix. The authors then apply a k-medoids clustering approach to divide the body of malware samples into clusters that combine samples with similar behavioral profiles. Once training is complete, a new and unknown sample is assigned to the cluster whose cluster medoid is closest to the sample. [6]

ISSN : 2455-9679
Impact Factor : 3.340

IV. CONCLUSION

This paper has described the threats that computer viruses to research and development multi-user computer systems; it has attempted to tie those programs with other, usually simpler, programs that can have equally devastating effects. Many author were investigated how anti-virus software analyzes the infected file and shows pro-missing approach for malware detection in the future. To combat the never ending virus generation, the anti-virus software company should work closely with researchers to find potential approach that both work efficiency and accuracy.

REFERENCES

[1] P. Denning, "The Science of Computing: Computer Viruses," American Scientist 76(3) (May 1988) pp. 236-238.

[2] B. Randell, P. Lee, and P. Treleaven, "Reliability Issues in Computing System Design," Computing Surveys 10(2) (June 1978) pp. 167-196.

[3]] K. Thompson, "Reflections on Trusting Trust," Communications of the ACM 27(8) (Aug. 1984) pp. 761-763.

[4] F. Cohen, "Computer Viruses: Theory and Experiments," Computers and Security 6(1) (Feb. 1987) pp. 22-35.

[5] T. Duff, "Experiences with Viruses on UNIX Systems," Computing Systems 2(2) (Spring 1989) pp. 155-172.

[6] Lee, T. and Mody, J. J. 2006. Behavioral classification. In European Institute for Computer Antivirus Research Conference (EICAR).

[7] Chen, H., Dean, D., and Wagner, D. 2004. Model Checking One Million Lines of C Code. In 11th Annual Network and Distributed System Security Symposium (NDSS04).

[8] Chen, H. and Wagner, D. 2002. MOPS: an infrastructure for examining security properties of software. In Proceedings of the 9th ACM conference on Computer and communications security (CCS). 235 – 244.

[9] Chen, X., Andersen, J., Mao, Z., Bailey, M., and Nazario, J. 2008.