**The Business Need for Security**

# Metamorphic Worms: Can they remain hidden?

**Reethi Kotti**

*Submitted to:* **Dr. Themis A. Papageorge**

*Course:* **Foundations of Information Assurance**

# CONTENTS


I. Worms
   A. What are Worms ……..... 1
   B. Few Popular Worms ……..… 2
   C. Propagation of Worms ……..…. 3
   D. Worm Signatures and Detection Strategies ……..... 5

II. Metamorphic Worms
   A. Introduction …….. 6
   B. Polymorphic vs. Metamorphic Worms ……..... 6
   C. Challenges faced during Detection …….. 7
   D. Detection Strategies ……. 8

III. Result
   A. Metamorphic Engines …….. 9
   B. Research Answer …… 14

IV. Conclusion ……... 15

V. References …… 16

# ILLUSTRATIONS


| | | |
|---|---|---|
| **Figure 1** | Topological Scanning | 5 |
| **Figure 2** | Random Scanning | 5 |
| **Figure 3** | General Obfuscation | 9 |
| **Figure 4** | Entry Point Obfuscation example | 10 |
| **Figure 5** | Illustration of a fragmented section of code | 11 |
| **Figure 6** | Illustration of the execution flow of a fragmented program | 12 |
| **Figure 7** | Zmist engine decompiling a PE file and integrating codes. | 13 |

# I. WORMS

## *A. What are worms*

Worms can be thought of us as malicious codes that exploit the vulnerabilities of a system. However there is one feature that makes them unique, it is their ability to propagate over a network. Though a worm sounds similar to a virus, they vary greatly in the method they employ to keep themselves alive in a network. Both worms and viruses propagate from one system to the other; however, a virus needs to attach itself to a file, data or executable, to be able to move over the network. On the other hand, a worm can propagate autonomously, i.e., without any assistance of external software. Also worm nodes might at times communicate with each other or with a central site, but a virus doesn't communicate with any external system.

A worm is generally broken down into five basic components; they are reconnaissance, attack, communication, command and intelligence components. The reconnaissance component is responsible for discovering new nodes that can be infected by the worms known methods. The attack component is what launches an attack on the discovered vulnerable nodes while the communication component is responsible for the communication that happens between the compromised nodes or with a central system. The command component provides the interface, which is required for sending out commands to the compromised nodes. Finally, the intelligence component is the one that has all the information that is required to establish communication between the various compromised nodes. This information can include location of the nodes and their characteristics.

To better understand a worm's propagation strategy explained later in this paper, it is a prerequisite to understand the different types of worms out there. They are broadly classified into scan-based and topology-based worms. A scan-based worm is one that propagates by probing the entire IPv4 space to find vulnerable hosts, while a topology-based worm relies on the information contained in the victims system to find its next target. The latter kind is more reliable as the number of guesses made in the former type is on the higher side.

While there are numerous papers on worms, most deal with their propagation techniques or introduce methods for overcoming their disastrous impacts. Few introduce methods of detecting them just in time to save the network from the otherwise arising repercussions.

In this paper, I attempt to delve deeper into a specific kind of worm not researched widely, known as the metamorphic worm. These worms are insidious, in that they can morph their behavior or code as they propagate. I intend to better understand these worms and in the process come up with an answer to the research question, can a metamorphic worm be made to go into a suspended state as a part of one of its iterations to avoid detection

## *B. Few popular worms*

In this section, few famous worms will be introduced which are particularly known for the havoc they wrecked.

1. **Morris worm, 1988**
   Robert Tappan Morris as a part of his research project created this worm, which escaped and crashed the Internet. The fallout because of this was large enough to cause the creation of CERT [23].
   It propagated by connecting to the Sendmail daemon and issuing the debug command to enter the debug mode. Once in the debug mode, it could pipe data through a shell and execute itself using the local C compiler and linker. It also exploited the finger daemon, creating a buffer overflow to invoke the bin/sh from where the worm could compile itself.

2. **Code Red v1 and v2, 2001**
   This worm attacked the web servers that were designed to be accessible around the world so it could bypass typical firewall installations. The attack was against the Microsoft IIS Web servers; buffer overflow was used to force the system to execute arbitrary actions based on the attacker. Once on the system, the worm multiplied forming 100 copies of itself, the first 99 threads began scanning to discover new hosts while the $100^{th}$ thread checked the locale of the infected server. The DoS attack against the site *http://www.whitehouse.gov* took place once every month. While propagating, the worm looked for two files on the systems, first is the file stating that the worm has been there before and the other is the date. This worm spread faster than the Morris worm as the Internet was booming by 2001. There were certain vulnerabilities in this worm that caused its demise. The random number generator employed by the worm used a constant seed value due to which it always generated the same random numbers. The other factor was that, an administrator thus fooling the worm could manually create the file it looked for.

3. **Nimda, 2001**
   This worm was first of its kind as it used multiple propagation mechanisms, due to which it became widespread very quickly [23]. The mechanisms it used included web server attacks where it performed scanning looking for IIS servers it could exploit. The other mechanism used was through electronic mails, where it exploited a known vulnerability in the Microsoft e-mail client. It also spread via open Windows networking file shares, infecting the file system on the target computer. The last propagation scheme was by attacking web clients where it uploaded an exploit to the home page of an infected site.

4. **Win32/Fujacks-AU, 2006**
   This is a worm with a backdoor functionality for the Windows platform [10]. This

worm tries to modify files on local and remote drives with extensions- HTML, HTM, PHP, ASP, JSP, and ASPX. These files are modified by appending an iframe which redirects the browser to a remote website from where the worm tries to copy additional Trojans to the system or update itself. It also copies itself to mapped network drives with the filename setup.exe, and drops the file autorun.inf, which runs automatically. It also attempts to delete files with a GHO extension. It was famous for the panda icon it left behind after infection.

5. **Worm: Win32/Vundo.A, 2009**
   Vundo family is known for displaying pop-ups that are usually related to fake antivirus software [11]. It spreads by copying itself to mapped network drives. It may also prevent security features and processes from functioning properly. It downloads itself onto the system as a DLL file, which then creates a mutex, which ensures that only one instance of the worm is running at a time. The worm connects to the servers like 85.12.43.102, pancolp.com and exficale.com from where it installs updates and downloads other malware. It also disables the phishing filter in Internet explorer 7 by modifying the registry. This worm, as suggested by the name, is exclusive for Windows platform.

## *C. Propagation of worms*

Hosts in the network are in three states during the propagation of worms- susceptible, infectious and removed. Susceptible is when the host is vulnerable to the infection, infectious host is one that as been infected and can infect others and a removed host is dead or has been removed so it cannot be infected.

Although many techniques have been introduced to detect and prevent worms, they still pose a significant threat. There are three main reasons behind it; first, the advancement in technology and networks has caused the propagation of worms to overtake the speed of human-mediated responses. Secondly, worms can propagate through the whole network in a matter of few seconds establishing themselves on every system with a particular vulnerability. This voracious propagation speed is one of the main reasons why worms are still being coded. Thirdly, worms being produced today are complicated and more efficient.

In this section we will deal with a variety of propagation techniques employed by worms.

i. **Scan-based techniques**- this is the easiest way to propagate hence is widely used. In this method, a set of IP addresses is scanned to identify vulnerable hosts [2].

   - Random Scanning: in this method, a target is randomly selected. Hence the

whole topology is fully connected and each edge has identical infection probability.

a) Uniform Scanning- in this method, the worm selects its victim without any preference. Code Red version 1 and 2, and scanner worms employed this technique.
b) Hit-list Scanning- in this method, hosts on the hit list are first infected, and then random scanning is employed to identify other vulnerable hosts. Flash worm employed this technique. Targets on the hit-list are infected with great speed as time is not wasted on scanning. Hence this technique speeds up the initial propagation of the worm.
c) Routable Scanning- here, scanning is performed only on the targets in the routable address space and not the entire IPv4 address space.

- Localized Scanning: instead of selecting targets at random, the closest addresses are chosen for scanning. This leads to a fully connected topology, where hosts within the same group can infect each other with the same infection probability while the infection probability for hosts in other groups is different.
    a) Local Preference Scanning- vulnerable hosts aren't distributed evenly in real world. Hence the probability of finding a vulnerable host is higher if you scan a densely populated area. This is employed here, so an IP address close to a propagation source with a higher probability is chosen over one farther away.
    b) Local Preference Sequential Scanning- in this method, scanning is done is an order, it beings at the starting IP address. If a host closer by is chosen with high probability than a worm farther away, then it's called local preferential sequential scanning. In this you're more likely to repeat the same propagation sequence, which results in wastage of infection power.
    c) Selective Scanning- this approach is employed if the attacker wants to attack a particular address area. Hence the worm will scan and infect the vulnerable hosts only in the target domain.

**ii.** **Topology-based Techniques**- In these techniques [2], the worm propagates using topological neighbors, so it utilizes the information in the victim's machine. This technique is considered more efficient than scan-based techniques as the latter makes a lot of guesses to propagate. Human interference is involved here, for instance, if you consider an email worm, it will become widespread only when an email user opens the worm email attachment.

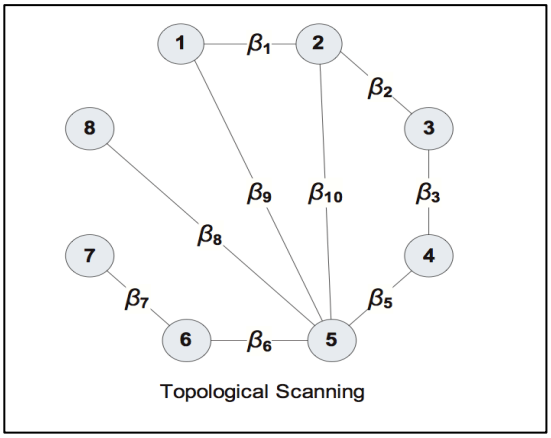


**Figure 1. Topological Scanning**

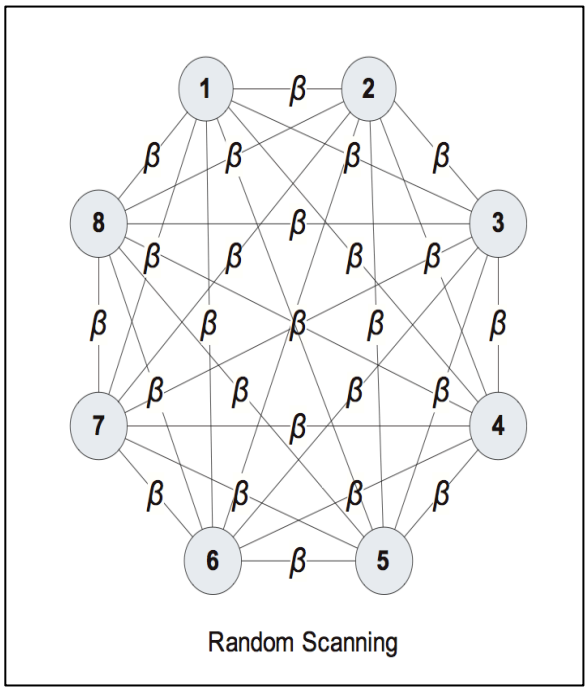


**Figure 2. Random Scanning**

*Source*: Both the illustrations are adopted from Yini Wang, Sheng Wen and Yang Xiang: "Modeling the Propagation of Worms in Networks: A survey". *IEEE Communications Surveys and Tutorials* 16, no.2, 2014.

## *D. Worm signatures and detection Strategies*

The easiest way to detect worms is by understanding its characteristics. All worms have at least few overlapping behavior patterns, like an increase in the network traffic.

Hence in this section, I introduce three detection strategies- traffic analysis, signature based detection and using honeypots and dark network monitors.

- **Traffic analysis:** We can detect a worm by performing traffic analysis, that is, by observing three characteristics specific to a worm-infected network. The first would be to analyze the volume of traffic on the network; this increases as more nodes in the network get infected.
  The next way is to monitor the type of scans happening on the network. Active infected hosts perform a number of scans to identify vulnerable systems. These scans can be tracked using measurement and monitoring tools, which can lead us to the infected hosts.
  Every host on the network has well-defined characteristics, so deviations in these characteristics can also help detect compromised hosts.

- **HoneyPots and Dark (Black Hole) Network Monitors:** Here you set up stations that will monitor worm activity passively. So we use honeynets and dark network monitors that probe and log whatever they see [4].
  A honeynet is a network of honeypots, which lure the attackers in by appearing vulnerable. While dark network monitors watch unused network segments for malicious traffic. These can be local or global unused networks.
  Honeypots provide access to a small data set while the dark network monitors will provide data generated from a large network space. Together these two tools can be used for identifying worms.

- **Signature-based Detection:** The underlying concept used here is pattern matching which employs a dictionary containing known bad signatures [24]. Three main types of signature analysis are used; first being network payload signatures where packet contents are compared. Next is logfile analysis, where you use application and system logs to perform the analysis. The last method uses file signatures, i.e., file payloads of worms and their executables.

# II. METAMORPHIC WORMS

## *A. Introduction*

A metamorphic worm is a worm that can reprogram itself. With each infection, it rewrites its code, making it appear different, but the main functionality of the worm doesn't change. This change of code is done using a metamorphic engine. This ability to morph itself makes detecting these worms harder.

Translating the original code into a temporary representation and then editing this, rewriting itself back to normal code is often performed to attain the morphing effect.

This characteristic gives metamorphic worms an edge, which is, they can remain undetectable with respect to static analysis. Therefore there is a good chance that the number of metamorphic worm attacks may keep increasing in the near future.

While a metamorphic virus can afford to have its metamorphic engine separated from the main virus body, like the NGVCK virus; a metamorphic worm would most likely have to carry its own metamorphic engine as it propagates across the network, unlike a virus. This need to carry its engine provides for complications as the morphing engine itself can act as a signature. Hence the metamorphic worm must morph its own engine every time it morphs its code. This imposes restrictions on the structure of the morphing engine.

## *B. Polymorphic worms vs. Metamorphic worms*

Polymorphic worms and metamorphic worms are used synonymously but they vary due to their respective unique engines. A polymorphic engine can transform a program into a version consisting of different code but having the same functionality. Encryption is generally employed here; encrypting the payload with different keys can generate many worm variations. A decryption module has to be prepended before the payload [7].

A metamorphic engine aims to modify the malicious code itself. This can be done via renaming registers, transposing code blocks, and through instruction transformation. Here, actual transformations are performed on the code by the engine, for example [7],

```
B9 00 10 00 00          mov ecx, 1000h
Is transformed to,
B9 10 B2 00 3C          mov ecx, 3C00B210h
81 C1 F0 5D FF C3     add ecx, 0C3FF5DF0h; ecx = 1000h
```

## *C. Challenges faced during detection*

Most of the techniques employed to detect worms assume that the worm's behavior does not vary or remains constant. This assumption was made as most of the earlier worms followed a fixed attack pattern. They would begin by scanning for vulnerable hosts, upon finding one; they would propagate to it and compromise the system. Then use that node to repeat the same pattern again.

The process a general worm uses to compromise a system also remains the same. Hence if the same attack is spotted on multiple systems, it can be concluded that the same worm has infected these systems. Hence by finding ways to monitor the pre-established pattern, we can identify a worm-like activity. By paying closer attention to the victims, we can figure out if all the victims were compromised by the same worm-

same vulnerability, same method employed to exploit the discovered vulnerability are indicators that the attack was by the same worm.

But by the advent of polymorphic and metamorphic worms, these techniques have proven ineffective. This is primarily because these worms keep altering their malicious code; hence they don't possess a particular signature that can be tracked; so the deduction that the same worm has compromised all the affected victims cannot be made by conventional mechanisms. This would give the worm enough time to further spread through the network making it harder to control them.

## *D. Detection Strategies*

There are a number of papers that propose novel detection strategies for metamorphic and also polymorphic worms. But none have yet been established as standards. A few promising detection strategies found, will be elaborated in this section.

a. **Detection using Hidden Markov Model:** In this detection strategy, HMM is employed to identify metamorphic worms [22], [24]. The basic objective here is to train an HMM using opcodes extracted from worms belonging to a particular family. This trained HMM will then represent the statistical properties of that worm family. So it can be used to determine how 'close' a file is to the worm family that the HMM represents.
   A lot of research has been done on how to avoid detection using this model, and it has been found that by inserting dead code into the malicious code, making the file look closer to a benign file, a worm can avoid detection by this model. However this will work only when the amount of dead code added is more than 2.5 times the original malicious code size.

b. **Double honeypot system:** In this technique [8], two honeypots are employed; one being inbound while the other is outbound. The inbound system is configured to not make any connections to other systems. When a worm attacks the inbound system, it will be configured to scan other systems by the worm code, this will prove that the system has been compromised. This traffic will be directed to the outbound honeypot that will then analyze the attack pattern and worm's malicious code. A gate translator is employed on the router between the Internet and the network that will direct unwanted traffic to the inbound honeypot. This model can detect metamorphic worms in that it incorporates both signature and anomaly based techniques. Byte sequences can be captured using this technique, which will then be employed to come up with worm signatures.
   When code transposition is done, a lot of jump instructions are added to the code, these can be removed using executable-analysis techniques. Swapping of registers causes only minor changes in the code sequence.

# III. RESULT

The research question posed was regarding metamorphic worm's ability to avoid detection- can a metamorphic worm suspend itself into an inactive state to avoid detection.

After considerable amount of research, I identified that the Code Red worm was known to remain dormant on an infected system for approximately a month before attacking again. But it has to be noted that Code Red was a worm without the presence of a morphing engine.

A deeper understanding of the functionality of a morphing engine and the techniques employed by it is thus required to be able to answer the research question. The summary of the data collected on morphing engines is explained in the coming sub-section and an answer for the research question is presented in the sub-section following that.

## *A. Morphing engines*

As already noted, a metamorphic worm is capable of rewriting its code while maintaining the same functionality with each infection. This is achieved via a morphing engine that uses various methods to represent the same code in a number of ways. A short description of few such popular techniques will be discussed here.

- **General Obfuscation:** A very common technique employed by both polymorphic and metamorphic worms to hide from antivirus scanners. The methods employed include garbage code insertion, registry modification, code transformation etc. Here you try to manipulate the ordering of the code, replace a high-level instruction with a combination of low-level instructions, change method names in a program or try to modify the aggregation of control data. In garbage insertion, code with no effect like NOP functions or other complex codes are inserted to manipulate the byte sequence of the viral code. Fig. 3 briefly illustrates the mechanism of general obfuscation

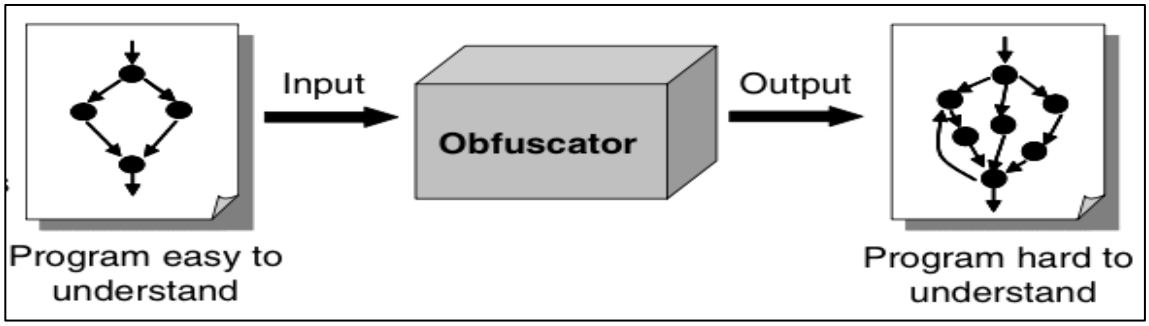


**Figure 3. General Obfuscation**

*Source*: The above illustration has been adopted from Yuichiro Kanzaki, Akito Monden, Masahide Nakamura and Ken-ichi Matsumoto: "Exploiting Self-Modification Mechanism for Program Protection", *27th Annual International Computer Software and Applications Conference*, IEEE 2003.

- **Entry Point Obfuscation (EPO):** In known heuristics, the antivirus programs or detection systems search for modified entry points of a target executable, as for a worm to acquire control, it needs to place itself in the line of execution. Under EPO, the worm, instead of modifying the entry point, places itself in the middle of the executable and using jump and call functions receives control from and transfers it back to the target executable. In Fig. 4 the entry point address is 401000, the malicious code is obfuscated using complex transformation EDI instructions. The metamorphic engine inserts these instructions such that no effect is seen on real file execution and since they are inserted after a PUSH function, we assume that the registers will be modified later on. Hence it's considerably difficult to capture these worms using a single signature.

- **Host Code Mutation:** Here the morphing engine transforms the code. In order to perform that, the engine first needs to implement a disassembly, which will parse the input code, and then the engine will transform this code to produce new code that will retain its functionality but look different. The engine apart from mutating its own code also modifies the code of the host.

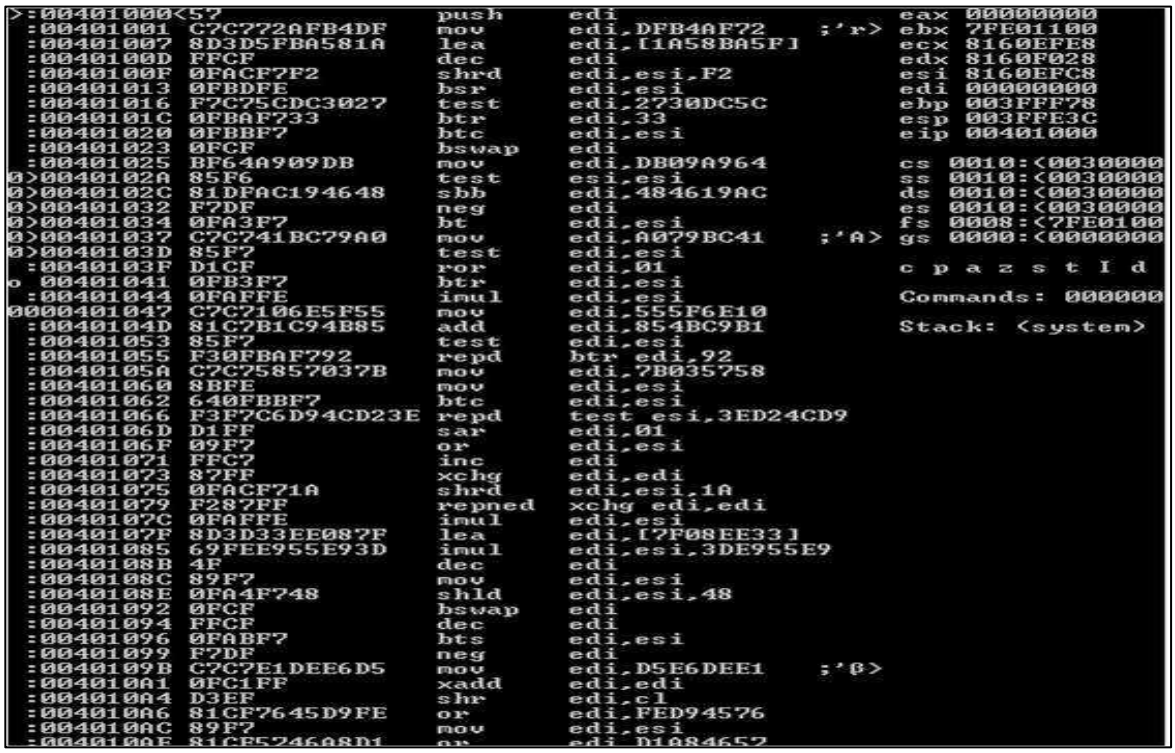


**Figure 4. Entry Point Obfuscation example**

*Source:* The above illustration has been adopted from Xufang Li, Peter K.K. Loh, Freddy Tan: "Mechanisms of Polymorphic and Metamorphic Viruses", *2011 European Intelligence and Security Informatics Conference*, IEEE 2011.

- **Program Fragmentation**: This technique [15] was first presented in a paper; in this technique, the metamorphic engine takes segments of code from various portions of the program and scatters them throughout the program. By removing segments of code, the spatial locality of the program is disturbed thus making the location of malicious code difficult. This is illustrated in Fig. 5. These sections of code are copied to random locations and a part of the memory at the original location is used up to place a procedure call or jump to the new location while the remainder of the memory is written over with random instructions. When the program executes and reaches the original location, the jump instruction gets executed which transfers the control to the new location, after execution of the instructions, the control returns back to the instruction following the original location thus skipping all the inserted random instructions. Fig. 6 illustrates this concept. A construct similar to a lookup table or function-manager is required to manage all the fragments; rather than directly jumping to the fragment, the program calls the function-manager which then transfers the control to the appropriate location. This technique is similar to subroutine reordering but has much smaller granularity as smaller segments of code can be placed into fragments, rather than entire subroutines.

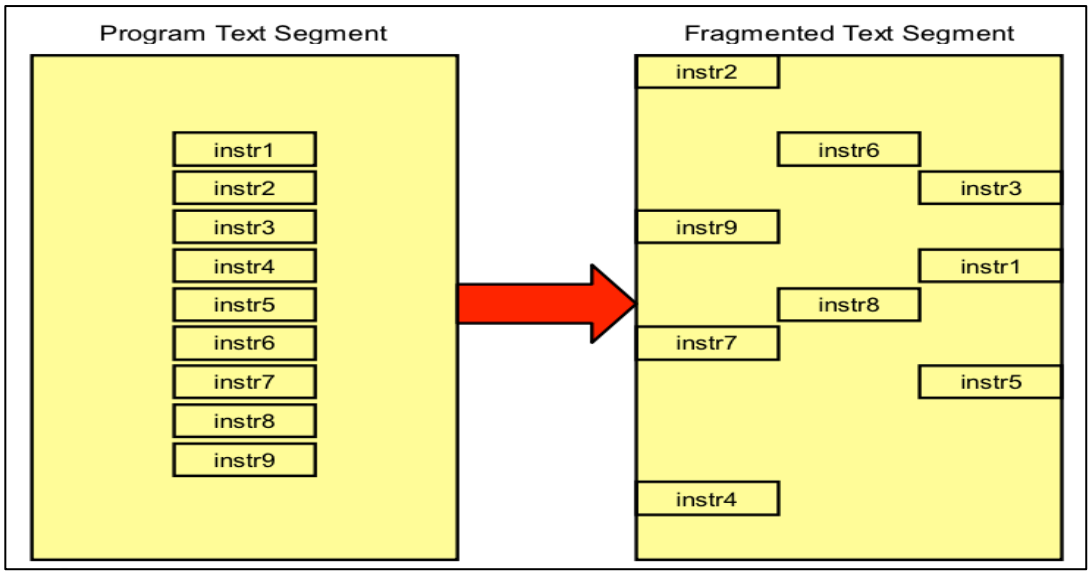


**Figure 5. Illustration of a fragmented section of code**

- **Code Transposition:** This method was used by Win32/Vundo worm [7], which was explained in earlier sections. Here the execution flow is altered at the instruction or module level using conditional and unconditional jumps.

Instructions whose functionality is essentially to do nothing are generated by the morphing engine and are inserted in gap places of the jump flows. The only thing that these instructions accomplish is modifying the original code flow.

For example,

Push ecx; Entry point
Jmp instruction 1
Instruction 1; Garbage code like NOP
Jmp instruction 2
Instruction 2; Garbage code
Entropy data; Packed or compressed or encrypted data

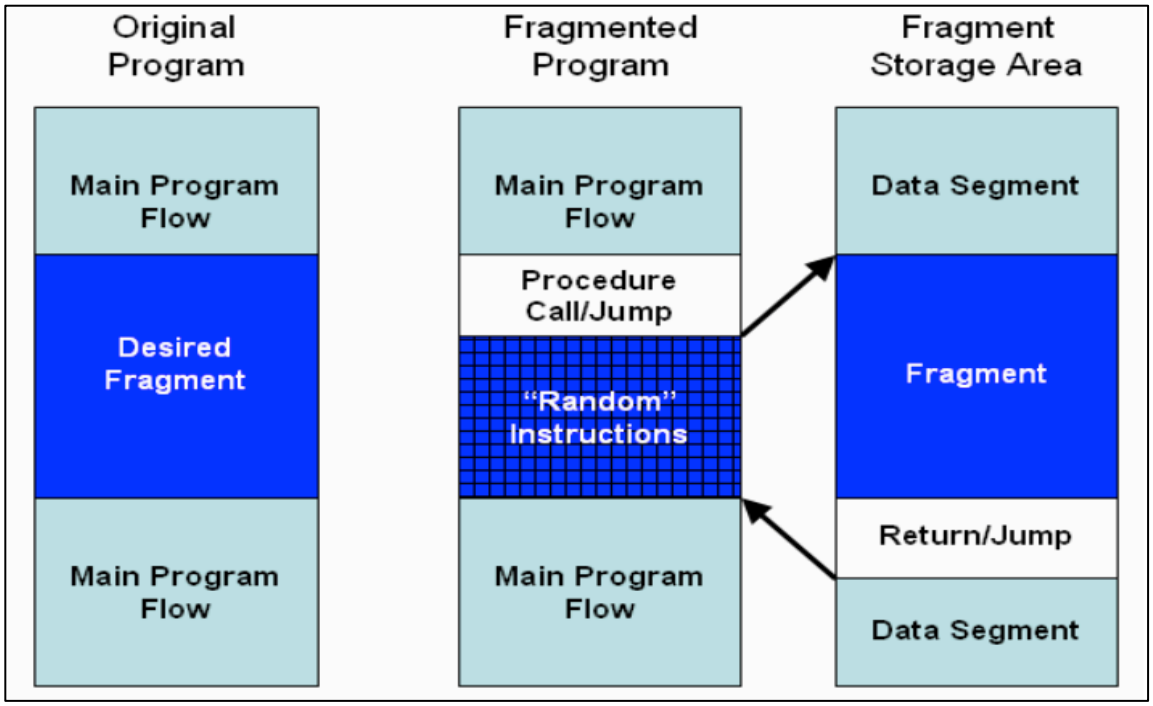


**Figure 6. Illustration of the execution flow of a fragmented program**

*Source*: The above two illustrations have been adopted from Bobby D. Birrer, Richard A. Raines, Rusty O. Baldwin, Barry E. Mullins, Robert W. Bennington: "Program Fragmentation as a Metamorphic Software Protection", *Third International Symposium of information Assurance and Security*, IEEE Computer Society, 2007.

- **Anti – Debugging:** The worm employs these techniques if it comes under the control of a debugger. Anti – Debugging is used to slow down the reverse engineering process by malicious codes, packers etc. In case of Win32.Evol, if the morphing engine finds a breakpoint on debugging, it would jump to a routine that would result in a crash.

  For instance, the following routine maybe employed which will eventually

result in a crash. Win32.Evol, a true metamorphic engine-powered malware, used this routine [7].

```
; START OF FUNCTION CHUNK
AntiDebug:
cmp byte pt [ebx+7], 0BFh; checks for kernel mode
jnz short ret_AntiDebug; jumps if not zero
mov ecx, 1000h; counter = 1000h
mov edi, 40000000h;
or edi 80000000h;
add edi, ecx; edi = C0001000h
rep stosd; bytes copied to edi
ret_AntiDebug:
retn; return
; END OF FUNCTION CHUNK; this will result in a crash since this routine wasn't called
```

- **Code Integration:** This technique was first seen in Win32/Zmist, written by researcher Peter Ferrie and Peter Szor. Zmist's engine first decompiles a Portable executable's code and then moves few code blocks out of the way and inserts itself there, then it rebuilds the code for execution. The methodology followed in code integration is shown in the Fig. 7.

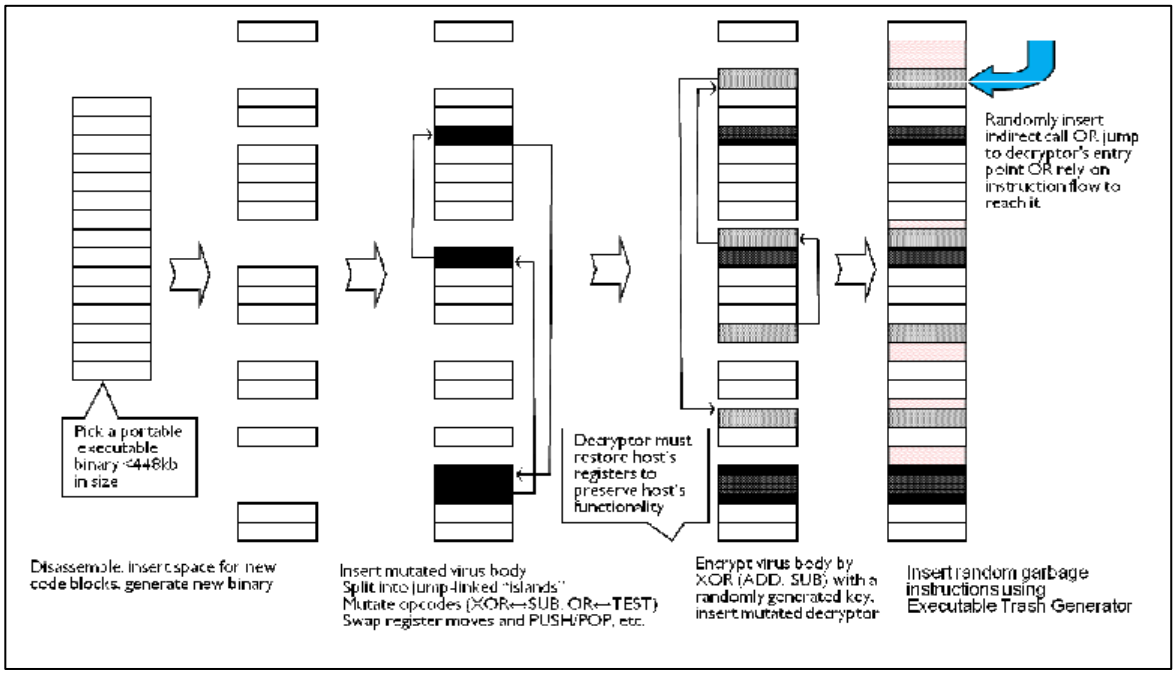


**Figure 7. Zmist engine decompiling a PE file and integrating codes.**

*Source:* The above two illustrations have been adopted from Xufang Li, Peter K.K. Loh, Freddy Tan: "Mechanisms of Polymorphic and Metamorphic Viruses", *2011 European Intelligence and Security Informatics Conference*, IEEE 2011.

## *B. Research Answer*

It has already been established that worms being suspended into an inactive state is possible. Worms like Code Red version 1 and 2, and Code Red II have accomplished this in the early 2000's.
A sample piece of code from Code Red worm [18],

```
push 6DDD00h; this is for 2 hours
call dword ptr[ebp-160h]; sleep
;
; this sleeps for 2 hours
```

As seen, this causes the worm to sleep for two hours before again continuing its function. By altering the value pushed, the sleep patterns can be changed. To illustrate this better, I now present another piece of code that causes the worm to sleep for 4.66 hours approximately.

```
push 1000000h;
; sleeps for around 4.66 hours
```

If a worm can be coded such that it sleeps randomly for irregular intervals time, there is a good chance that a signature for this worm would be difficult to generate, thus making the usage of signature-based detection strategies ineffective. For such a worm to be effective, a random number generator using a dynamic seed is critical. The random number generator should be used for determining the amount of time a worm would be suspended.

Since the morphing engine would be using techniques like garbage code insertion, code integration etc., the execution time of the worm on each system it infects would most probably be varied. Assuming that in at least four or five machines for every ten systems that a worm infects, the total execution time required is different, we can establish that the sleeping patterns will also vary. Also infected systems may differ in their configurations thus leading to varied execution times, thus complicating the construction of a signature for the worm.

But the catch here is that though a worm would be sleeping for irregular intervals of time, it is still sleeping for the same exact number of times. How this would affect the signature generation of the worm and if the number of times a worm sleeps can be modified with each infection by possibly employing a counter or a random generator is beyond the scope of this paper and hence is left for further advanced research.

# IV. CONCLUSION

Worms are one kind of malware that are not yet known to cause catastrophic results. This is probably because most of the worms generated so far have focused more on propagation rather than destroying or modifying data on the system they infect. This doesn't mean that they are not as harmful as viruses; it rather means that worms are yet in their early stages and have not been exploited by malware writers to their full potential. Hence there is a possibility that worms may soon become one of the most dreaded malware.

While it has been stated that worms are yet to be exploited, there have been improvements. For instance, metamorphic worms are a relatively new inclusion into the Worm family. Hence not a lot of study has been performed on this topic, which makes it a very fertile ground for research.

Metamorphic worms make detection harder by modifying themselves with each infection. The functionality of morphing engines has been presented in the earlier sections of this paper to better illustrate the uniqueness of metamorphic worms. Since metamorphic worms are still in their budding stages, the true capacity of these worms cannot be predicted.

In this paper, I tried to figure out the capability of a worm to suspend itself into a sleep state making it harder for the detection systems. After performing considerable amount of research, I came to a conclusion that suspension of the worm is very much possible and this ability can be included into the code that a morphing engine will modify. I also presented the likelihood of a worm that has the capability to sleep in combination with morphing its code and also offered a possibility for further research, which I intend to carry out in the future.

# REFERENCES


[1] Nazario, Jose. 2004. *Defense and Detection Strategies against: Internet Worms.* London: Artech House Boston.

[2] Wang, Yini, Wen, Sheng and Xiang, Yang. 2014. "Modeling the Propagation of Worms in Networks: A survey". *IEEE Communications Surveys and Tutorials* 16, no.2.

[3] Fan, Xiang and Xiang, Yang. 2010. "Modeling the Propagation of Peer-to-Peer Worms under Quarantine". *IEEE/IFIP Network Operations and Management Symposium – NOMS 2010: Short Papers.*

[4] Tang, Yong and Chen, Shigang. 2005. "Defending Against Internet Worms: A Signature-Based Approach". *24th Annual Joint Conference of IEEE Computer and Communications Societies, IEEE* 2.

[5] Lakhotia, Arun, Kapoor, Aditya and Uday, Eric. January, 2005. "Are Metamorphic Viruses Really Invincible".*Virus Bulletin.*

[6] Gebhart, Glenn. 2004. "Worm Propagation and Countermeasures". *SANS Institute, 2004.*

[7] Li, Xufang, Loh, Peter and Tan, Freddy. 2011. "Mechanisms of Polymorphic and Metamorphic Viruses". *IEEE Intelligence and Security Informatics Conference, 2011 European.*

[8] Mohammed M.Z.E., Mohssen, Chan, Anthony, Ventura Neco, Hashim, Mohsim and Amin, Izzeldin. 2009. "Polymorphic Worm Detection Using Double-Honeynet". *IEEE.*

[9] Maurer, Jon. 2003. "Internet Worms: Walking on Unstable Ground". *Sans Institute 2003.*

[10] "W32/Fujacks-AU – Viruses and Spyware". http://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32~Fujacks-AU/detailed-analysis.aspx. (accessed November 9, 2014).

[11] "Worm: Win32/Vundo.A". http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Worm%3aWin32%2fVundo.A (accessed November 9, 2014).

[12] "IDA – evol.idb". http://www.openrce.org/articles/files/evol_disasm.html. (accessed November 10, 2014).

[13] "The Viral Darwinism of W32.Evol- Collaborative RCE Knowledge Library". http://www.threatexpert.com/report.aspx?md5=2d2ab339bc44c6292108cb2594ae01ff. (accessed November 10, 2014).

[14] “ThreatExpert Report: W32.Fujacks.E, Worm.Win32.Fujacks.cv”. http://www.threatexpert.com/report.aspx?md5=2d2ab339bc44c6292108cb2594ae01ff. (accessed November 11, 2014).

[15] Birrer, Bobby, Raines, Richard, Baldwin O., Rusty Mullins, Barry and Bennington, Robert. 2007. “Program Fragmentation as a Metamorphic Software Protection”. *Third International Symposium on Information Assurance and Security.*

[16] Dolak, John. 2001. “The Code Red Worm”. *Information Security Reading Room SANS 2001.*

[17] “CAIDA Analysis of Code-Red” http://www.caida.org/research/security/code-red/. (accessed November 14, 2014)

[18] “Code-Red-Worm [source code]” http://indianrenegade.blogspot.com/2007/02/code-red-worm-source-code.html. (accessed November 14, 2014)

[19] Kanzaki, Yuichiro, Monden, Akito, Nakamura, Masahide and Matsumoto, Ken-ichi. 2003. “Exploiting Self-Modification Mechanism for Program Protection”, *27th Annual International Computer Software and Applications Conference*, IEEE 2003.

[20] Wong, Wing and Stamp, Mark. 2006. “Hunting for metamorphic engines”. *Springer-Verlag France 2006.*

[21] “How to generate random numbers in ASM - Programming and Coding - Tuts 4You” https://forum.tuts4you.com/topic/26304-how-to-generate-random-numbers-in-asm/ (accessed November 15, 2014)

[22] Attaluri, Srilatha, Mcghee, Scott and Stamp, Mark. 2009. “Profile hidden Markov models and metamorphic virus detection”, *Journal in Computer Virology 5, no.2 (May) : 151-169.*

[23] Syed, Farhan. “Understanding worms, their behavior and containing them”, *Project Report, University of Washington, St. Louis.* Available: *http://www.cse.wustl.edu/~jain/cse571-09/ftp/worms.pdf*

[24] Madenur Sridhara, Sudarshan. 2012. “Metamorphic Worm that carries its own Morphing Engine”. *Masters Thesis and Graduate Research, San Jose State University*.