# A STATE OF THE ART SURVEY ON POLYMORPHIC MALWARE ANALYSIS AND DETECTION TECHNIQUES

**Emmanuel Masabo[1], Kyanda Swaib Kaawaase[2], Julianne Sansa-Otim[3], John Ngubiri[4] and Damien Hanyurwimfura[5]**
[1,2,3,4]*College of Computing and Information Sciences, Makerere University, Uganda*
[5]*College of Science and Technology, University of Rwanda, Rwanda*

*Abstract*
***Nowadays, systems are under serious security threats caused by malicious software, commonly known as malware. Such malwares are sophisticatedly created with advanced techniques that make them hard to analyse and detect, thus causing a lot of damages. Polymorphism is one of the advanced techniques by which malware change their identity on each time they attack. This paper presents a detailed systematic and critical review that explores the available literature, and outlines the research efforts that have been made in relation to polymorphic malware analysis and their detection.***

***Keywords:***
***Polymorphic Malware, Static Analysis, Dynamic Analysis, Machine Learning, Malware Detection***

## 1. INTRODUCTION

System's security is a major concern in today's computing environment. There are more powerful threats appearing on daily basis. According to Symantec report [1], there were almost a million newly created threats that were injected into the wild on daily basis in 2014. The anti-malware industry and malware creators try to outsmart each other by constantly developing more advanced techniques. Early malware was spread through file transfers using floppy disks and removable disks. Once a user opened an infected file on a clean computer, the system could be infected [2]. Later malware was much more complex and were mainly spread from the Internet. Currently, malware is even more complex and highly sophisticated in terms of attacking and evading detection [3]. They are able to change their identity at every fresh attack without changing the body of the virus [4], thus exhibiting the same behaviour. They are also capable of mutating into an infinite number of new variants [5]. Many researchers proposed a number of useful approaches to deal with polymorphic malware as described in survey articles [6] [7]. Despite all the efforts undertaken to contain the problem of polymorphic malware, it remains challenging [8] [9] and requires more research endeavours. This study gives the state of the art and outlines the research efforts in relation to the techniques used in analysis and detection of polymorphic malware. Several techniques exist and each has its own strengths and weaknesses. To accomplish the objectives of this study, the author searched journals, conferences and search engines. The primary contributions of this paper are:

- To provide an analytical review of the most successful later works on polymorphic malware detection.
- To retrieve the gaps that require further research.
- To assess the adequacy of the most popular tools used to analyze polymorphic malware.
- To identify the most promising studies that can serve as the point of reference for the future research.
- To provide recommendations and directions for future research.

The rest of this paper is organized as follows: section 2 discusses the concepts, structure and working mechanisms of polymorphic malware. Section 3 portraits the techniques used in analyzing polymorphic malware. Section 4 discusses polymorphic malware detection techniques. Finally, the overall conclusion and recommendations are discussed in section 6.

## 2. MALWARE STRUCTURE

### 2.1 GENERAL MALWARE

Malware is a kind of malicious code which is destructive. It is just like any other regular software. However, it has a harmful intent such as denial of service, attempt to confidentiality, information stealing or abuse of integrity [10]. To achieve its objectives, a malware can perform many operations.

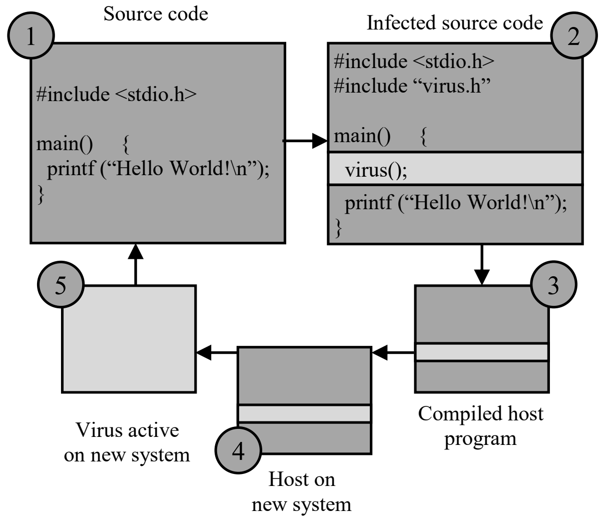


Fig.1. Infection by a Code Virus

Some of them are described as follows: file activity (open, read, delete, modify, move), Registry activity (open, create, delete, modify, move, query, close), Service activity (open, start, create, delete, modify), Mutex (create, delete), Processes (start, terminate), Runtime DLLs, Network activity (TCP, UDP, DNS, HTTP). Malware types range from simple to more complex ones such as polymorphic malware. Malware can perform a self-execution locally or can be controlled remotely via the Internet. Malware families include but not limited to spyware, adware, cookies, trapdoor, Trojan horse, sniffers, spam, botnet, logic bomb, Worm, Virus, key-loggers, ransomware, backdoors, adware, spyware. Fig.1 below shows the example of a simple virus infection.

## 2.2 POLYMORPHIC MALWARE

The first polymorphic malware was developed by Mark Washbrun in 1990 and was called 1260 virus [11]. Polymorphism is a stealth technique used by malware to create an unlimited number of new different malware variants [11] [10] [12] in order to harden analysis and detection. Their code is constantly changing at any infection [13]. In general, polymorphic malware has invariant bytes that are constant across all their created instances as well as variant bytes that change values at every infection [7] [14]. The infection process is described in Fig.2.

In order to obstruct analysis, polymorphic malware use code obfuscation techniques [15] such as packing, encryption and junk code insertion or substitution [2] [16] [17] [11]. Packed files have different hash values. Packing is mainly used by malware to harden the reverse engineering, to harden a dynamic debugging process, to reduce execution file size, to harden static malware disassembly, to bypass malware detection and to defeat malware researchers [2]. The structure of a packed file is shown in Fig.2.

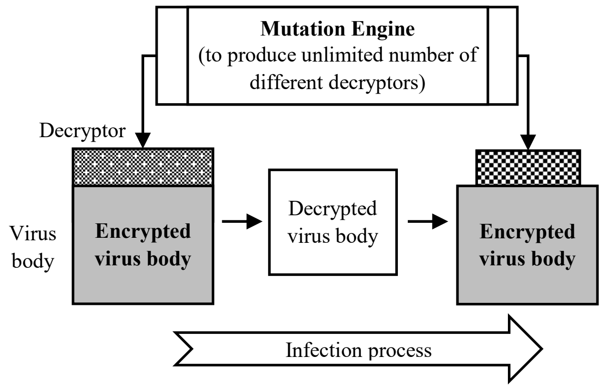


Fig.2. Polymorphic malware infection process [11]

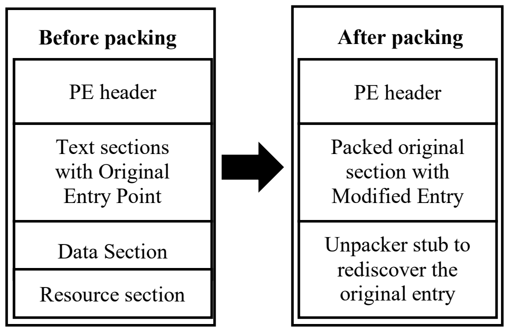

Before packing
PE header
Text sections with Original Entry Point
Data Section
Resource section
After packing
PE header
Packed original section with Modified Entry
Unpacker stub to rediscover the original entry

Fig.3. Packing and unpacking process

## 2.3 COMPONENTS OF A POLYMORPHIC MALWARE

There are two categories that predominantly characterize the segments of a polymorphic malware, namely the invariant and variant bytes [7] [4].

### *2.3.1 Invariant Bytes:*

Protocol framing is in charge of branching down the execution path of the code with a string that remains unchanged across all instances of polymorphic malware. Return address or function pointers are values that are used in overwriting a jump target to redirect execution. Exploit code is a set of invariant bytes in charge of malicious activities. It ensures that malicious behaviours are identical in all newly created malware variants [18].

### *2.3.2 Variant Bytes:*

Encrypted code or payload contains malicious codes that keep changing at every infection. Decryption module is in charge of decrypting the encrypted payload and control is passed to malicious code to start execution. These modules are obfuscated in different variants of polymorphic malware. Decryption key is required to decrypt the payload because multiple keys are generated by a polymorphic engine in order to allow the creation of multiple malware instances.

## 2.4 OBFUSCATION TECHNIQUES

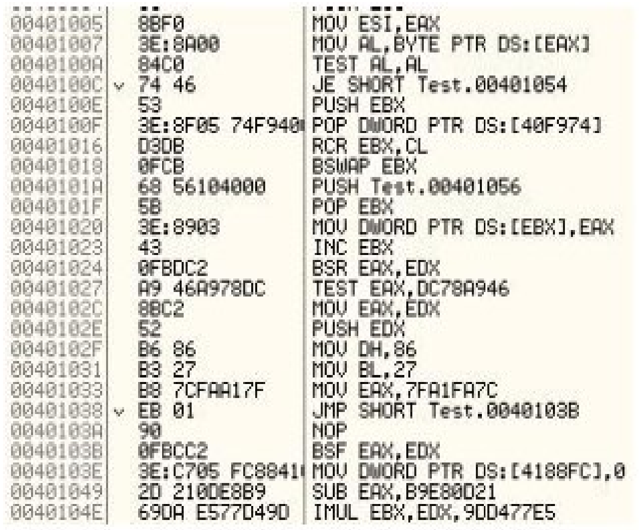
Obfuscation makes malware harder to detect [19]. It transforms a malicious code to a new different version while keeping the same functionality [11] [19]. Originally, this technique was used for protecting software developer's intellectual property. Later, it has been used by malware creators to neutralize detection and analysis [11], [19], [20]. The obfuscated code is different from the original one and is hard to understand while trying to conceal malware internal purposes [21]. Obfuscation adds less important instructions or garbage to an existing code to change its structure or appearance and yet retain the same behaviour [19] [22]. The binary sequence of a malicious code is modified without affecting the original functionality. Obfuscation techniques include instruction replacement, instruction permutation, variable or register substitution, junk or dead code insertion and code transposition [19].

### *2.4.1 Dead Code Insertion:*

The Fig.4 and Fig.5 respectively show the part of the original code and dead code insertion examples. To inject a dead code, a Non-operation (NOP) code is injected into the original code. A NOP sled is a long sequence of instructions that are often included in the shellcode as part of an exploit in order to increase the probability of an exploit to be successful. Code analysis task becomes very complicated when the code is nested as shown in Table.1. Instructions in Table.1 can change the status flag registers, but no change is made on the value of the operand register [11]. It is clear that adding a zero to a register, or assigning a register value to itself doesn't have any effect on the execution results [11].

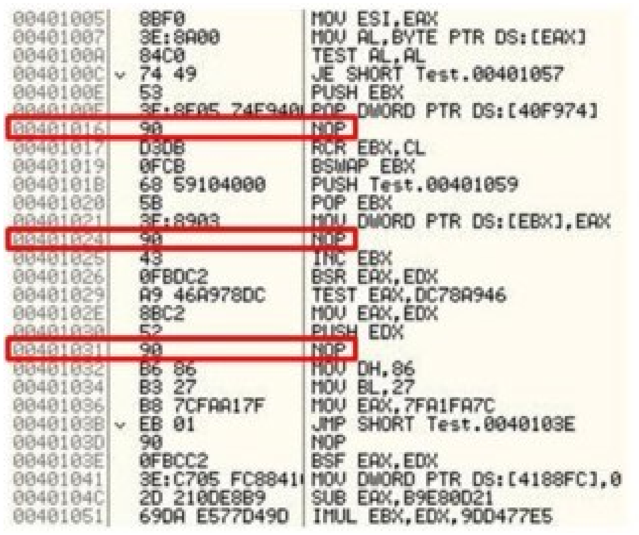
Table.1. Non-operation code injection [11]

| Instruction | Operation |
|---|---|
| ADD Reg, 0 | Reg <- Reg + 0 |
| MOV Reg, Reg | Reg <- Reg |
| OR Reg, 0 | Reg <- Reg\|0 |
| AND Reg, -1 | Reg <- Reg &-1 |



```
00401005   8BF0             MOV ESI,EAX
00401007   3E:8A00          MOV AL,BYTE PTR DS:[EAX]
0040100A   84C0             TEST AL,AL
0040100C v 74 46            JE SHORT Test.00401054
0040100E   53               PUSH EBX
0040100F   3E:8F05 74F940   POP DWORD PTR DS:[40F974]
00401016   D3DB             RCR EBX,CL
00401018   0FCB             BSWAP EBX
0040101A   68 56104000      PUSH Test.00401056
0040101F   5B               POP EBX
00401020   3E:8903          MOV DWORD PTR DS:[EBX],EAX
00401023   43               INC EBX
00401024   0FBDC2           BSR EAX,EDX
00401027   A9 46A978DC      TEST EAX,DC78A946
0040102C   8BC2             MOV EAX,EDX
0040102E   52               PUSH EDX
0040102F   B6 86            MOV DH,86
00401031   B3 27            MOV BL,27
00401033   B8 7CFAA17F      MOV EAX,7FA1FA7C
00401038 v EB 01            JMP SHORT Test.0040103B
0040103A   90               NOP
0040103B   0FBCC2           BSF EAX,EDX
0040103E   3E:C705 FC8841   MOV DWORD PTR DS:[4188FC],0
00401049   2D 210DE8B9      SUB EAX,B9E80D21
0040104E   69DA E577D49D    IMUL EBX,EDX,9DD477E5
```

Fig.4. Original code [19]



```
00401005   8BF0             MOV ESI,EAX
00401007   3E:8A00          MOV AL,BYTE PTR DS:[EAX]
0040100A   84C0             TEST AL,AL
0040100C v 74 49            JE SHORT Test.00401057
0040100E   53               PUSH EBX
0040100F   3E:8F05 74F940   POP DWORD PTR DS:[40F974]
00401016   90               NOP
00401017   D3DB             RCR EBX,CL
00401019   0FCB             BSWAP EBX
0040101B   68 59104000      PUSH Test.00401059
00401020   5B               POP EBX
00401021   3E:8903          MOV DWORD PTR DS:[EBX],EAX
00401024   90               NOP
00401025   43               INC EBX
00401026   0FBDC2           BSR EAX,EDX
00401029   A9 46A978DC      TEST EAX,DC78A946
0040102E   8BC2             MOV EAX,EDX
00401030   52               PUSH EDX
00401031   90               NOP
00401032   B6 86            MOV DH,86
00401034   B3 27            MOV BL,27
00401036   B8 7CFAA17F      MOV EAX,7FA1FA7C
0040103B v EB 01            JMP SHORT Test.0040103E
0040103D   90               NOP
0040103E   0FBCC2           BSF EAX,EDX
00401041   3E:C705 FC8841   MOV DWORD PTR DS:[4188FC],0
0040104C   2D 210DE8B9      SUB EAX,B9E80D21
00401051   69DA E577D49D    IMUL EBX,EDX,9DD477E5
```

Fig.5. Dead code insertion [19]

### 2.4.2 *Register Exchanging:*

This technique involves switching registers or memory variables of different malware variants [22] [19] while keeping the identical behavior [20] and changing the binary sequence of the code. Two forms of W95.Regswap malware are shown in Fig.6 and Fig.7 respectively. It is clearly shown that the functionality is the same, but registers have been interchanged [23] [24]. This can easily defeat signature-based detection systems. The Table.2 also shows how polymorphic techniques are implemented.

| Binary Code Sequence | Assembly Code | |
|---|---|---|
| 5A | pop | edx |
| BF04000000 | mov | edi,0004h |
| 8BF5 | mov | esi,ebp |
| B80C000000 | mov | eax.000Ch |
| 81C288000000 | add | edx,0088h |
| 8B1A | add | ebx,[edx] |
| 899C8618110000 | mov | [esi+eax*4+00001118],ebx |

Fig.6. Variant 1 of W95.Regswap malware [11]

| Binary Code Sequence | Assembly Code | |
|---|---|---|
| 58 | pop | eax |
| BB04000000 | mov | ebx,0004h |
| 8BD5 | mov | edx,ebp |
| BF0C000000 | mov | edi.000ch |
| 81C088000000 | add | eax,0088h |
| 8B30 | mov | esi,[eax] |
| 89B4BA18110000 | mov | [edx+edi*4+00001118],esi |

Fig.7. Variant 2 of W95.Regswap malware [11]

### 2.4.3 *Instruction Replacement/Substitution:*

Instructions are replaced by equivalent ones as illustrated in the following example, where all given instructions are equal as they set the register eax to 0.

MOV eax, 0

XOR eax,eax

AND eax, 0

SUB eax, eax

### 2.4.4 *Instruction Permutation/Reordering:*

The sequence of instructions is reordered randomly with the purpose of making the code binary sequence look different in multiple instances of the same malware [11] [19]. This reordering does not affect the functionality of the malware. This technique is able to generate *n*! Different instances, where *n* is the number of subroutines. An example of instructions permutation is given in Table.3.

### 2.4.5 *Code Transposition:*

This technique reorders or shifts the binary code sequences and uses unconditional or conditional branches to recover the original program execution flow [20], [11] [19] as shown in Fig.8.

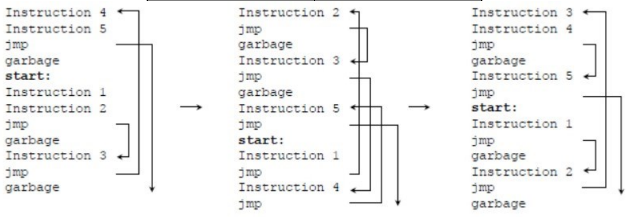
Table.2. Polymorphic techniques

| Original code |
|---|
| #include<stdio.h><br>int main()<br>{<br>int *a*,*b*,*c*;<br>*c*=1;<br>*a*=0;<br>*b*=6; |

| |
|---|
| while (*c*<*b*)<br>{<br>*a*= *a*+*b*;<br>*c*++;<br>}<br>} |
| **Variable renaming** |
| #include<stdio.h><br>int main()<br>{<br>int *m*,*n*,*p*;<br>*p*=1;<br>*m*=0;<br>*n*=6;<br>while(*p*<*n*)<br>{<br>*m*=*m*+*n*;<br>*p* = *p*+1;<br>}<br>} |
| **Statement reordering** |
| #include<stdio.h><br>int main()<br>{<br>int *a*,*b*,*c*;<br>*b*=6;<br>*a*=0;<br>*c*=1;<br>while(*c*<*n*)<br>{<br>*a*= *a*+*b*;<br>*c*++;<br>}<br>} |
| **Statement replacing** |
| #include<stdio.h><br>int main()<br>{<br>int *a*,*b*,*c*;<br>*c*=1*1;<br>*a*=0*c;<br>*b*=6/*c*;<br>while(*c*<*b*)<br>{<br>*a*= *a*+*b*;<br>*c*++; }<br>} |
| **Junk code insertion** |
| #include<stdio.h><br>int main()<br>{<br>int *a*,*b*,*c*;<br>*c*=1;<br>*a*=0;<br>*b*=6;<br>bool *h*=true;<br>if(*h*)<br>{<br>while(*c*<*b*)<br>{<br>*a*= *a*+*b*;<br>*c*++;<br>}<br>}<br>} |
| **Spaghetti code** |
| #include<stdio.h><br>int main()<br>{<br>int *a*,*b*,*c*;<br>goto *T*;<br>*N*: while(*c*<*b*)<br>{<br>*a*= *a*+*b*;<br>*c*++;<br>}<br>goto *F*;<br>*T*:*c*=1;<br>*a*=0;<br>*b*=6;<br>goto *N*;<br>*F*:<br>} |

Table.3. Instruction Reordering

| Order 1 | Order 2 |
|---|---|
| mov eax, 0F<br>push eax<br>add esi, ebx | add esi, ebx<br>mov eax, 0F<br>push ecx |



```
Instruction 4
Instruction 5
jmp
garbage
start:
Instruction 1
Instruction 2
jmp
garbage
Instruction 3
jmp
garbage
```

```
Instruction 2
jmp
garbage
Instruction 3
jmp
garbage
Instruction 5
jmp
start:
Instruction 1
jmp
Instruction 4
jmp
```

```
Instruction 3
Instruction 4
jmp
garbage
Instruction 5
jmp
start:
Instruction 1
jmp
garbage
Instruction 2
jmp
garbage
```

Fig.8. Binary code shifting [11]

### *2.4.6 Code Integration:*

This technique is very sophisticated as the malware first decompiles the target program into very small objects and secondly adds itself to them, then finally reassembles the integrated code to create a new file [19][21].

### *2.4.7 Virtualization Obfuscation:*

Instructions are virtualized to hide from the analysis. The malware includes a virtual machine module that is used to interpret the virtualized code [11].

## 3. POLYMORPHIC MALWARE ANALYSIS

Analysis is a process that allows the analyst to get a clear picture of the malware structure and functionality [25]. During analysis, different key features are extracted [26] and provide knowledge about the functionality of a malware. Informal categories of analysis include vulnerability analysis [27], source code analysis [6] [27], and behavioral analysis [6]. Similarity analysis [21] [28] is undertaken to ensure that a malware is a variant of an existing one. Analysis task is very important and is the first thing to be done prior to developing reliable detection systems. In general, two main analysis techniques exist, namely Static analysis and Dynamic analysis [6].

## 3.1 STATIC ANALYSIS

Static analysis is a process whereby information about the malicious program is extracted without executing it [20], [29], [30]. This makes static analysis safer than dynamic analysis as malware is not run [31]. It is classified in two categories such as basic static analysis and advanced static analysis. The basic static analysis gives basic information about the malicious program such as its version, file format, any suspicious imports, etc. Basic static analysis is straightforward and quick, but not very effective as it can miss important details [32]. Advanced static analysis deals with code/structure analysis where the knowledge of assembly language, compiler code, and Operating system concepts is required [32]. Malware functionality is analysed by inspecting the internal code of the malware. This analysis is able to provide information regarding malware identity, passwords, libraries, URL, programming language, etc. Function routines and mutants can be identified [33]. With advanced static analysis, the code can be disassembled and decompiled [29]. However, static analysis is unable to handle packing and obfuscation. It reveals the presence of packing, but the binary needs to be unpacked for the success of static analysis [13]. In addition to what has been explained above, the following information can also be revealed by static analysis [6], [25], [34], [35]:

- **File fingerprinting:** The cryptographic hash code (eg. md5) is computed to know if a binary has not been modified.
- **Hash coded string extraction:** This process helps to extract human-readable strings embedded in the compiled binary. With this information, a conclusion is drawn about some functionality of the binary. Imported and exported functions are revealed.
- **File format:** The metadata of a file format is investigated to extract useful information such as file type, file format and compilation time.
- **Antivirus Scanning:** if a binary is already known, it will be detected by one or more anti-malware programs.
- **Packer Detection:** due to obfuscation structure (e.g. encryption and compression) of that malware, suitable tools such as PEID [32] and Detect It Easy (DIE) can identify packer and compiler information.
- **Disassembly:** This process consists of reverse engineering machine code to assembly language which is understood by a human. Tools like IDA pro [32] can be used. The generated assembly code helps the analyst inspect further the logic of the code and gather more information about the intent of the malware. OllyBdg [32] tool is used in this process.

Static analysis has two main advantages. First, it is safe during the analysis process, because a malware doesn't have to be executed. Secondly, it provides deeper information about execution paths of a malware. In addition to that, the static analysis also has some disadvantages such as requiring much experience, consuming a lot of time, not being able to handle packed files, and producing some ambiguous results, in case of binaries that use self-modifying techniques to indirect jump instructions [34].

## 3.2 DYNAMIC ANALYSIS

Dynamic analysis is the process of analysing a malicious program by first executing and monitoring its runtime functionalities [20]. Malicious behaviors are monitored and logged [6], [25], [34]. During this process, the malware unpacks itself and the changes it makes on the system are also observed. Dynamic analysis is undertaken in a virtual environment in order to ensure full protection of the host machine [15], [29], [30], [36]. The basic dynamic analysis consists of observing the basic behaviors of a malware such as process creation, file activities or registry activities [32]. On the other hand, the advanced dynamic analysis makes a profound examination of the internal state of a running malicious program. It uses advanced debugging techniques to single step through the malicious code and makes a deep internal inspection to get more comprehensive information on the malicious behaviors [32]. The code is analysed at runtime and any hidden code through packing is revealed [13]. The identity of a malware is dynamically identified. Function calls, parameter analysis, and information flow are all visualized. Dynamic link libraries (dll), processes and file activities are revealed [29], [37]. Dynamic analysis helps detect polymorphic mutants as well as packing and obfuscation attributes, [14]. Memory analysis can be performed [30]. The interaction between malware, file system, processes, and network is inspected [30]. However, dynamic analysis is computationally expensive and requires a lot of system resources [15] [35].

Advantages of dynamic analysis include the investigation of live malware behaviours at runtime, handling packed files, and automated analysis of large malware corpus. The disadvantages of Dynamic analysis include the possibility of missing out some execution paths when a malware being monitored becomes dormant, risk of spreading the infection by a network capable malware from the virtual environment to the host system; and the impossibility to monitor the malware when it has the capabilities of refusing to execute when it is run in a virtual environment.

In addition to seeing runtime behaviors of a malware, dynamic analysis can allow an analyst get the difference between two system states. An initial state is recorded before running the malware and another state is recorded after malware execution. A comparison will be done and differences highlighted to assess the impact of the malware on the system [38].

Both static and dynamic analyses are essential techniques needed for a successful malware investigation task towards developing detection approaches. Table.4 draws the comparison between both techniques.

Table.4. Comparison between both Static and Dynamic analyses

| Static analysis | Dynamic analysis |
|---|---|
| • A binary is not executed prior to analysis<br>• Doesn't necessarily need a virtual environment | • A binary must be executed prior to analysis |

| | |
|---|---|
| • Can't handle packed binaries before manually being unpacked<br>• It's hard, but can reveal almost the full code coverage or global view of a binary<br>• Low false positive rate | • It needs a virtual environment set up before analysis<br>• Packed binaries are automatically unpacked [11]<br>• Reveals the only path execution of modules that are running<br>• High false positive rate |

## 3.3 MALWARE ANALYSIS TOOLS

### *3.3.1 Static Analysis Tools:*

**PEiD**: This tool is used to detect the type of the packer or the compiler used to build an application. This can help the analyst identify the suitable program to unpack the binary. Entropy and checksum can also be calculated.

**UPX**: This is an ultimate packer for executables that helps in compressing and packing or decompressing and unpacking executables. Packing process makes malware hard to analyze and detect. It shrinks the size of malware binaries. The packed program hides the original data and creates an unpacking stub that is called by the operating system when a malware attacks. Unpacking process helps in rediscovering the original executable and transfers execution to the original entry point. This tool can unpack malware packed by UPX or other packing tools as well.

**IDA Pro**: This is an interactive disassembler that is used to reverse engineer malware binaries and create maps of execution as well as discovering and analysing vulnerabilities. Tasks done by IDA include function discovery, stack analysis, local variable identification, and much more. This is the choice of many malware analysts. Additional key features of this tool include code disassembly, discovering obfuscation traces, function parameters analysis, function calls analysis, operation codes analysis, strings analysis, persistence tracking, privilege escalation attack detection, DLLs injection analysis, Runtime DLLs, Process replacement functionalities, code noise patterns, Rootkit functionalities, Anti debug/analysis constructs, Kernel mode activity, User mode activity, user-defined functions, and attacker identification.

### *3.3.2 Dynamic Analysis Tools:*

**Process Monitor**: This tool provides real-time monitoring of file system, processes, registry when the malware is running and visualizes all the events as they are happening. Running processes, system calls, file system, Network activity, registry activity, API calls, Mutex and Self-modifying code traces can all be detected.

**Dependency Walker**: This tool helps explore Dynamic Link Libraries (DLLs), imported functions, exported functions and monitor interactivity between running processes and DLLs.

**Regshot**: This tool takes system snapshots before and after the malware is executed and creates a log of registry manipulations. Both snapshots are then compared to observe registry modifications. Registry modifications and system changes give a clue of malware behaviors.

**ApateDNS**: This tool helps the analyst to capture DNS requests made by malware without necessarily being connected to the Internet. It imitates DNS responses to a given IP address. It can display all received results in a hexadecimal format.

**Wireshark**: This is a packet sniffing program that helps to understand how malware performs network communications. It can intercept and log the traffic that passes over the network when the malware is running. Wireshark provides packet details which enable the analyst to do the in-depth packet streams analysis.

**INetSim**: This is a tool suitable for simulating common Internet services. It can provide fake services that allow the analyst to analyze the network behaviors of a malware. Services that are emulated by InetSim are but not limited to: HTTP, HTTPS, FTP, IRC, DNS and SMTP. InetSim can also record all inbound connections and requests which are very useful to see if a malware is connected to any service and what requests it is making.

### *3.3.3 Sandboxes:*

Sandboxes are tools that are able to analyze a malware using both on static and dynamic analysis techniques. They provide a detailed report at the end of the analysis. Malware are executed within the sandbox and all side effects remain within the sandbox system without affecting the host system.

- **Anubis**: This is an automated malware analysis tool that helps monitor API functions, system service calls, function parameters, etc.
- **CwSandbox**: This is an automated malware analysis tool that helps monitor API calls, system calls, Registry manipulations tracking, Network traces, Operating System interactions
- **GFI sandbox**: Automated malware analysis tool to monitor API calls.
- **JoeBox sandbox**: This is an automated malware analysis tool. Its key features include file system analysis, registry activity analysis, system calls analysis, rules generation, memory dump analysis, packing analysis and strings analysis.
- **Cuckoo sandbox**: This is an automated malware analysis tool capable of analyzing File activity, registry, system calls, API, memory dumps, packing, network activity, etc Different files formats can be analyzed as well as URLs. This tool also has advanced memory analysis capabilities that help in discovering deeper hidden malware functionalities.

## 4. POLYMORPHIC MALWARE DETECTION TECHNIQUES

Malware detection is simply a technique valuable for identifying malware in all targets [39], either computers, applications or cyber-physical devices [40]. Polymorphic malware has many variants of the same functionality as discussed in previous sections. In order to detect them, similarity analysis is undertaken.

## 4.1 SEQUENCE CLASSIFICATION DETECTION [41]

Drew et al. [41] proposed a detection approach based on sequence classification methods. Based on biological gene sequence mutations. Authors use the same concept to assess similarities in mutations of malware. They used a tool developed for gene classification called The Super Threaded Reference-Free Alignment-Free Nsequence Decoder (STRAND) for classifying polymorphic malware.

**Evaluation**: This method was evaluated on a 500 GB malware dataset provided by Kaggle-Microsoft Malware Classification Challenge (BIG 2015). Features were developed for STRAND classifier from given sample bytes, where hex values were considered and missing values removed. This created a single string/strand sequence containing only all hex contents extracted from malware file. Sequences of words of length k or k-mers were generated by strand. Word length of 10 characters and 2400 minihash values were used. Jaccard similarities were computed. Training time was less than 7 hours and strand classification accuracy was 95%, of which 9 classes of polymorphic malware in the given set were detected.

## 4.2 SEQUENCE-BASED CLASSIFICATION AND ENSEMBLES DETECTION [42]

In their expanded version of work [41], Drew et al. in [42] developed an approach that detects polymorphic malware using sequence classification methods and ensembles. STRAND algorithm was used for the classification task.

**Evaluation**: In this work, they also used Kaggle dataset of 500 GB from Microsoft 2015 challenge. They used bytes (.bytes) data and assembly data (.asm) features as provided by Microsoft. Ensembles were created by using both the asm and bytes models from Strand. The minihash technique was used to detect similarity. Minihash scores of created models were computed and the overall detection accuracy became 98%.

## 4.3 DROPPED FILES BASED POLYMORPHIC MALWARE DETECTION [17]

Selamat et al. [17] proposed an easy detection technique for detecting polymorphic malware based on dropped files.

**Evaluation**: To evaluate their technique, authors in [17] executed malware in a virtual environment. They used process explorer to observe malware behaviors. Three different types of malware pythium2.exe, kax.exe, ieg.exe were used in the experiment. Each was executed twice. For the first and second executions, each malware could create a child process. It was observed that those created subprocesses during first and second executions were different. The conclusion was drawn that this is also a good indicator of polymorphic behavior.

## 4.4 TOPOLOGICAL BASED FEATURE EXTRACTION DETECTION [8]

Fraley et al. [8] proposed a detection approach that utilizes topological feature extraction using static and dynamic analysis techniques. The proposed method relies on data mining techniques as a means to achieve classification scalability.

**Evaluation**: This method uses IDA pro and cuckoo sandbox for feature extraction. Experiments were conducted on data set of 3637 samples of which 2400 were clean, 800 were malicious and 437 were unlabeled. They used function call graphs to trace instruction patterns. Belief propagation was used to uncover the properties of malicious files. Twenty features (12 structural and 8 behavioral) were collected and used to make a complete feature set. Few examples of extracted features include MOV, ADD, LEA, file size, and compiler type. The created feature set was converted into the ARFF data format compatible with the Weka data mining tool. Ensemble bagging algorithms were used in Weka for classification and cross-validation with 10 folds was used for validation. The overall accuracy was 99.9 % where low false negative is 0.001

## 4.5 STRING MATCHING AND SUBSTITUTION MATRICES [5]

Naidu et al. [5] proposed a detection method that involved the extraction of syntactic structures of a malware from its hex code. It helped in identifying the piece of code that is malicious and the variant type of the malware. The Smith-Waterman algorithm was used to identify variants.

**Evaluation**: Experiments were done using two variants of JS.cassandra malware. Hex dumps were extracted using sigtool from ClamAV. Hex bytes were then converted to binary code and from binary to DNA sequences. The DNA sequences were input into JAligner tool where Smith-Waterman algorithm (SWA) was implemented. Common substrings or patterns were extracted Pairwise local alignment (PLA) was performed using SWA with different substitution matrices. Only substrings with the highest percentage of identities and similarities were extracted after PLA. 161 sub-strings were retained through 6 substitution matrices. These 161 substrings have been considered as meta-signatures and used to detect all known polymorphic variants of JS.Cassandra. T-Coffee tool was used to perform multiple sequence alignment on the generated meta-signatures and generated consensuses. Rules were generated using a data mining classification algorithm called PRISM and this allowed the generation of 47 super signature substrings. Therefore, those 47 super signatures and 161 meta signatures were converted back to hexadecimal and tested against JS.Cassandra. This method was able to detect 100% of JS.Cassandra malware and all its known variants.

## 4.6 CLIENT-SERVER BASED DETECTION [2]

Harmonen in [2] proposed a client-server architecture for identifying polymorphic malware. This method has two advantages: storing remotely virus information databases without allowing individual client updates and protecting the client application as well.

**Evaluation**: The client application is deployed on the target system and keeps monitoring the system of interest. Suspicious files are investigated. File hash and file metadata information are sent to the server. The server compares the received data with the one in its database. When multiple hash values are received from different clients and the corresponding metadata is the same, the file is identified as a polymorphic malware. The server is able to determine if the file is benign, known polymorphic malware or a new variant.

## 4.7 VIRAL POLYMORPHIC MALWARE DETECTION [9]

Naidu et al. [9] proposed a polymorphic malware detection approach that is based on automatic signature extraction.

Needleman-Wunsch and Smith-Waterman algorithms are used for developing a detection mechanism.

**Evaluation**: This method is built on top of same author's previous work [5] which is discussed above. Needleman-Wunsch algorithm was used together with Smith-Waterman algorithm. JS.Cassandra and W32.Kitti polymorphic malware were used in experiments. Hex dumps were extracted and converted to DNA. Pairwise sequence alignments were performed, meta and super signatures were generated. Finally, this method was able to detect 100% of JS.Cassandra and W32.Kitti known polymorphic variants.

### 4.8 HYBRID CLUSTERING DETECTION APPROACH [43]

Sharma et al. [43] proposed an approach that uses signatures and pattern matching to detect polymorphic malware.

**Evaluation**: Experiments were done using tools such as SciPy, Numpy and Python. A python module called pydasm is used to extract features and its report recorded. Opcodes are extracted from the given instructions and are used for further processing. KNN algorithm is implemented and the method was able to detect polymorphic malware. The detection accuracy was not provided.

### 4.9 COMBINED TOKEN EXTRACTION AND SEQUENCE ALIGNMENT DETECTION [18]

Eskandari et al. [18] proposed a technique that combines two approaches, namely token extraction and multiple sequence alignment for achieving a high accuracy and noise tolerance flexibility.

**Evaluation**: To evaluate their method, the authors used DARPA 1999 intrusion detection system dataset. They extracted tokens from suspicious flows and eliminated irrelevant parts of suspicious flow. They generated Simple regular expression (SRE) signatures and applied multiple sequence alignment. Tests were done on DARPA 1999 intrusion detection system dataset, polymorphic versions of CodeRed II, Apache-Knacker, ATPhttpd and BIND-TSIG exploit malware. This method was able to analyze inherent similarities of samples, thus detecting successfully polymorphic malware.

### 4.10 BEHAVIORAL BASED SEQUENTIAL PATTERNS [20]

Ahmadi et al. [20] proposed a malicious files detection approach that is based on behavioral sequential patterns. API calls behavior features were extracted using dynamic analysis and API calls log was generated. Initial dataset was then created from log data by only considering the repetitive patterns. Feature selection was conducted using Fisher score algorithm. Support Vector Machines (SVM) and decision tree algorithms were used for malware classification. This method was evaluated on a sample of 806 malware and 306 benign files. A detection accuracy of 95% was successfully achieved.

### 4.11 ZERO DAY HYBRID DETECTION [3]

Kaur et al. [3] developed a hybrid anomaly and signature detection system was used to detect zero-day polymorphic worms in an active network flow.

**Evaluation**: The system architecture has three components such as Suspected Traffic Filter (STF), Zero attack evaluation (ZAE) and Signature generator (SG). Suspicious traffic is collected using STF where known malware are blocked and logged. Zero-day attacks are redirected to the honeypot for further analysis. STF module is composed of two components: Honeynet and IDS/IPS to capture and compare network the traffic flow. The Longest Common Prefix (LCP) algorithm is used in the comparison process. When IDS/IPS is found to have ignored an attack that was logged by honeynet, it means that it's a new unknown attack. Analyses are done by ZAE component by which malicious strings are extracted such as NOP sleds, decryptor, shellcodes and return addresses. The SG module generates content based signatures by using invariant bytes found in a polymorphic malware. The method has been tested on a sample of 15435 packets of which 734 were polymorphic. The detection rate was 96% with almost zero positive rates.

### 4.12 BEHAVIOR ANALYSIS OF MALWARE USING MACHINE LEARNING [15]

Arshi et al. [15] proposed a behavioral detection approach based on machine learning. They focused on malware classification and clustering. 1270 malware samples of different format pdf, executables, HTML, zipped and jpeg were analyzed. Logic Model Tree and K-Means algorithms have been used for the task of classification and clustering respectively. The results show that 18% of analyzed malware were embedded with networking capabilities to connect to the outer world, while 82% aimed to corrupt the system locally or network resources. Malware have also been grouped successfully according to their file format types.

## 5. DISCUSSIONS

This research explored the literature about polymorphic malware analysis, detection as well as involved techniques. The findings show that a great work has been done in previous research efforts. This problem has been widely addressed using machine learning techniques. However, some studies also tried to address it using other techniques used normally in DNA sequence such as multiple sequence alignment, Needleman-Wunsh, Smith-waterman, Strand gene sequence classifier, etc. Some studies use more than one algorithm to build models and select the most reliable one. The Fig.9 highlights the algorithms used by selected detection techniques.

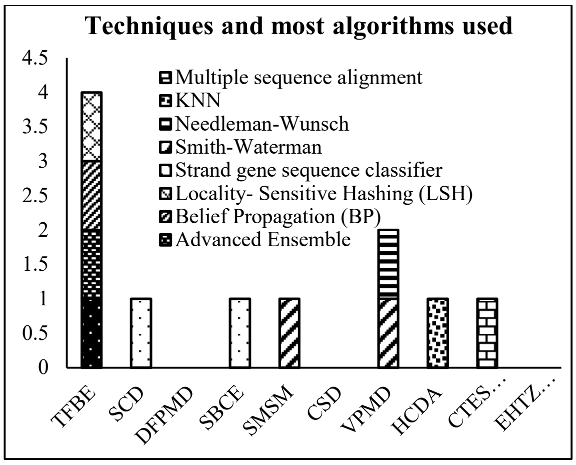


Fig.9. Algorithms used by selected techniques

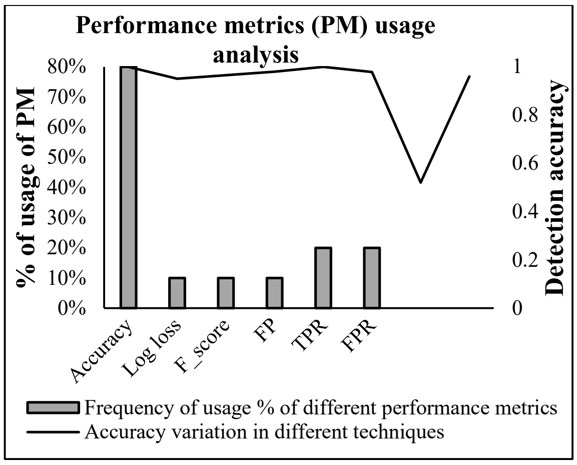


Fig.11. Performances of selected techniques

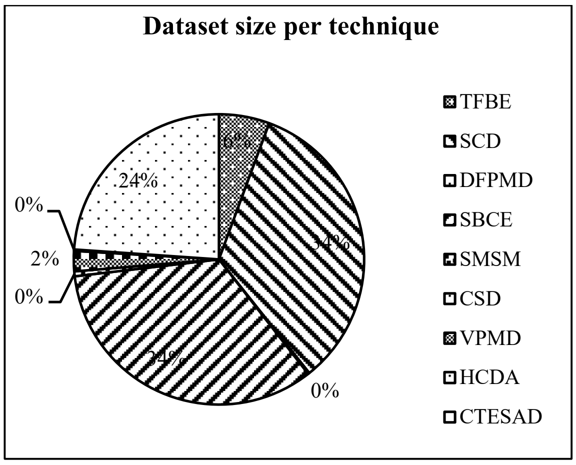


Fig.10. Number of dataset samples of the selected techniques

Table.5. Comparison of selected detection techniques

| Technique | Code | Contributions | Limitations |
|---|---|---|---|
| Topological features based extraction [8] | TFBE | A novel approach that extracts, analyses and combines multiple high factors to detect polymorphic malware. Faster and accurate for small samples | Need to be evaluated on a larger dataset. |
| Sequence classification detection [41] | SCD | A novel approach to detect polymorphic malware by using biological gene sequences and STRAND classifier. Can work on a larger sample | Considers statically extracted features only. Behavioural features were not considered. |
| Dropped files based Polymorphic Malware Detection [17] | DFPMD | Proposed a simple but fast technique of identifying polymorphic malware using dropped files. | False positives may incur, just in case where the genuine file has the same features. |
| Sequence-based classification and ensembles detection [42] | SBCE | A novel approach that extends [30] to detect polymorphic malware by using biological gene sequences and STRAND classifier. It uses (.bytes) and .asm files and can work on a larger sample. | Considers statically extracted features only. Behavioural features not considered. False positives may incur just in case where the genuine file has the same features |
| String matching and substitution matrices [5] | SMSM | Proposed a novel detection method that extracts of syntactic structures from hex code and implements Smith-Waterman algorithm to identify variants. Accurate on small samples. | Need to be evaluated on a larger dataset. Considers statically extracted features only. Behavioural features not considered |

| | | | |
|---|---|---|---|
| Client-server based detection [2] | CSD | Novel client-server approach where virus information is stored on a remote database and does not allow individual client updates | Serious problems can occur when the Internet is not available. |
| Viral polymorphic malware detection [9] | VPMD | Novel polymorphic malware detection approach that is based on automatic signature extraction and uses Needleman-Wunsch and Smith Waterman algorithms for detection mechanism. | Needs to be evaluated on a larger dataset.<br>Considers statically extracted features only. Behavioural features not considered.<br>Considers only known polymorphic variants of an existing malware |
| Hybrid clustering detection approach [43] | HCDA | Detection method that uses signatures and pattern matching to detect polymorphic malware. | Need to provide detection accuracy. Behavioural analysis needs to be considered |
| Combined Token Extraction and Sequence Alignment detection [18] | CTESAD | Novel signature-based technique that combines token extraction and multiple sequences alignment for achieving a high accuracy and noise tolerance flexibility.<br>It is fast and noise tolerant | Low detection accuracy. Behavioural analysis needs to be considered |
| Efficient hybrid technique for detecting zero-day polymorphic worms [3] | EHTZDD | Early detection and containment of zero-day polymorphic malware. Automatic signature generation | Behavioural features not considered |

Table.6. Data sources of selected techniques

| Techniques | Dataset source | Number of samples | Number of families | Malware only | Malware and benign |
|---|---|---|---|---|---|
| TFBE | ClamAV, VirusTotal, VirusShare and Contagio | 3637 | - | No | Yes |
| SCD | Microsoft kaggle malware dataset 500GB | 21741 | 9 | Yes | No |
| DFPMD | Second Part to Hell | 100 | 2 | Yes | No |
| SBCE | Microsoft kaggle malware dataset 500GB | 21741 | 9 | Yes | No |
| SMSM | Second Part to Hell | 352 | 1 | yes | No |
| VPMD | Second Part to Hell | 1457 | 2 | Yes | No |
| CTESAD | DARPA 1999 Intrusion Detection Evaluation Datasets | 100 | 4 | Yes | No |
| EHTZDD | honeypots | 15435 | - | No | yes |

Table.7. Evaluation methods and performance metrics used by the selected techniques

| Techniques | Val CV | Test split | Test Sample | Accuracy | Log loss | F score | FP | FN | ROC | TPR | FPR |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TFBE | yes | - | - | 0.9999 | - | - | 0.0001 | | | 0.997 | 0.003 |
| SCD | yes | - | - | 95% | 0.222864 | - | - | - | - | - | - |
| DFPMD | - | - | - | - | - | - | - | - | - | - | - |
| SBCE | yes | - | - | 98% | - | - | - | - | - | - | - |
| SMSM | - | - | - | 100% | - | - | - | - | - | - | - |
| CSD | - | - | - | - | - | - | - | - | - | - | - |
| VPMD | - | - | - | 100% | - | - | - | - | - | - | - |
| HCDA | - | - | - | 97.83% | - | 95% | - | - | - | - | - |
| CTESAD | - | - | - | 52% | - | - | - | - | - | - | - |
| EHTZDD | - | - | - | 96% | - | - | - | - | - | 0.961 | 0.06 |

Table.8. Algorithms and analysis methods used by the selected techniques

| **Techniques** | **Static Analysis** | **Dynamic Analysis** | **Algorithms** |
|---|---|---|---|
| TFBE | Yes | Yes | -Meta bagging<br>-Advanced Ensemble Classification<br>-Belief Propagation (BP),<br>-Locality-Sensitive Hashing (LSH) |
| SCD | Yes | No | -Strand gene sequence classifier<br>-Jaccard Similarity |
| DFPMD | | Yes | - |
| SBCE | Yes | | -Strand gene sequence classifier |
| SMSM | Yes | No | -Smith-Waterman algorithm |
| VPMD | Yes | No | -Needleman-Wunsch Smith-Waterman |
| HCDA | yes | No | -KNN |
| CTESAD | No | No | -Multiple sequence alignment |
| EHTZDD | Yes | No | - |

To evaluate the performance of their approaches, most researchers used accuracy as the main performance metric. Accuracies achieved were in the range of 90% to 100% as shown by Fig.11. Some other studies used also other performance metrics such as log loss, TPR, FPR, F-score, etc. as shown in Fig.11. Detection rates of 100% are subject of further exploration because it can simply be an over-fitting situation. This can happen when the dataset used is very small or the quality of the dataset itself. The Fig.10 shows the datasets sizes used in the presented techniques. Another observation is that during analysis, some studies extracted either static features or dynamic features, while others considered a hybrid of both types of features. Either can lead to good results in terms of performance metrics. However, an optimal detection system could use hybrid features and achieve better results by minimizing the disadvantages of static and dynamic analysis while maximizing their advantages.

## 6. CONCLUSIONS

In this research, we have provided a detailed discussion on polymorphic malware characteristics and attacking strategies. We have specifically discussed tools and techniques used in analysis and detection of polymorphic malware. We provided an adequate comparison through which weaknesses and strengths of each techniques were discussed. We found that, there are much more sophisticated attacking mechanisms built in polymorphic malware, that negatively affect the performance of the currently available tools and strong defensive techniques. Therefore, research must be done continuously as a major way to find more adequate solutions to address this big problem. Based on this review, the following major gaps still have to be taken into consideration in order to develop more efficient detection solutions:

- There is a need for an improved feature engineering mechanism that could address efficient detection at a larger scale.
- There is a need to investigate the impact of a combination of analysis and detection techniques for the improvement of detection approaches.
- There must be an improvement on detection mechanisms to improve real-time detection before the system is infected.
- There is a need for an interactive coordination in terms of information sharing and automatic detection responsibilities among detection systems. This could be implemented in a multi-agent based detection approach.
- There is a need for dynamic vulnerability analysis capabilities implemented in detection systems. This will enable them to scan installed applications and report problems in time to the software owners. Therefore, this will limit the spreading of zero-day polymorphic malware if owners can quickly fix the bugs.

## ACKNOWLEDGMENT

We would like to thank the Mobility to Enhance Training of Engineering Graduates in Africa (METEGA) and the Regional Universities Forum for Capacity Building in Agriculture (RUFORUM) for supporting our research.

## REFERENCES

[1] Symantec, “Internet Security Threat Report”, Available at: https://www.itu.int/en/ITU-D/Cybersecurity/Documents/Symantec_annual_internet_threat_report_ITU2015.pdf, Accessed on 2015.

[2] T. Harmonen, “Identifying Polymorphic Malware,” US. Grant, 2014.

[3] R. Kaur and M. Singh, “Efficient Hybrid Technique for Detecting Zero-Day Polymorphic Worms”, *Proceedings of IEEE International Advance Computing Conference*, pp. 95-100, 2014.

[4] S. Paul and B.K. Mishra, “PolyS: Network-based Signature Generation for Zero-Day Polymorphic Worms”, *Proceedings of IEEE International Advance Computing Conference*, Vol. 6, No. 4, pp. 159-163, 2013.

[5] V. Naidu, “Using Different Substitution Matrices in a String- Matching Technique for Identifying Viral Polymorphic Malware Variants”, *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 2903-2910, 2016.

[6] R. Kaur and M. Singh, “A Survey on Zero-Day Polymorphic Worm Detection Techniques”, *IEEE Communications Surveys and Tutorials*, Vol. 16, No. 3, pp. 1520-1549, 2014.

[7] S. Paul and B.K. Mishra, “Survey of Polymorphic Worm Signatures”, *International Journal of u-and e-Service, Science and Technology*, Vol. 7, No. 3, pp. 129-150, 2014.

[8] J.B. Fraley and M. Figueroa, “Polymorphic Malware Detection using Topological Feature Extraction with Data Mining”, *Proceedings of IEEE SoutheastCon*, pp. 1-7, 2016.

[9] V. Naidu and A. Narayanan, “Needleman-Wunsch and Smith-Waterman Algorithms for Identifying Viral Polymorphic Malware Variants”, *Proceedings of IEEE* 14th

*International Conference on Dependable, Autonomic and Secure Computing*, pp. 326–333, 2016.

[10] I.A. Saeed, J.B. Campus, M.A. Selamat, M. Ali and M.A. Abuagoub, "A Survey on Malware and Malware Detection Systems", *International Journal of Computer Applications*, Vol. 67, No. 16, pp. 975-987, 2013.

[11] B. Rad, M. Masrom and S. Ibrahim, "Camouflage in Malware: from Encryption to Metamorphism", *International Journal of Computer Science and Network Security*, Vol. 12, No. 8, pp. 74-83, 2012.

[12] M. Chau, G. Alan Wang and H. Chen, "A Syntactic Approach for Detecting Viral Polymorphic Malware Variants", *Proceedings of Pacific-Asia Workshop on Intelligence and Security Informatics*, pp. 146-165, 2016.

[13] S. Cesare, Y. Xiang and W. Zhou, "Malwise-an Effective and Efficient Classification System for Packed and Polymorphic Malware", *IEEE Transactions on Computers*, Vol. 62, No. 6, pp. 1193-1206, 2013.

[14] G. Liang, J. Pang and C. Dai, "A Behavior-Based Malware Variant Classification Technique", *International Journal of Information and Education Technology*, Vol. 6, No. 4, pp. 291-295, 2016.

[15] D. Arshi and M. Singh, "Behavior Analysis of Malware using Machine Learning", *Proceedings of 8th International Conference on Contemporary Computing*, pp. 481-486, 2015.

[16] K. Rieck, P. Trinius, C. Willems and T. Holz, "Automatic Analysis of Malware Behavior using Machine Learning", *Journal of Computer Security*, Vol. 19, No. 4, pp. 639-668, 2011.

[17] N.S. Selamat, F. Hani, M. Ali and M. Science, "Polymorphic Malware Detection", *Proceedings of International Conference on IT Convergence and Security*, pp. 12-18, 2016.

[18] M. Eskandari, M.S. Razieh and A. Asadi, "Automatic Signature Generation for Polymorphic Worms by Combination of Token Extraction and Sequence Alignment Approaches", *Proceedings of IEEE 7th Conference on in Information and Knowledge Technology*, pp. 116-126, 2015.

[19] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey", *Proceedings of International Conference on Broadband, Wireless Computing Communication and Applications*, pp. 297-300, 2010.

[20] M. Ahmadi, A. Sami, H. Rahimi and B. Yadegari, "Malware Detection by Behavioural Sequential Patterns", *Computer Fraud and Security*, Vol. 2013, No. 8, pp. 11-19, 2013.

[21] D. Uppal, V. Mehra and V. Verma, "Basic Survey on Malware Analysis, Tools and Techniques", *International Journal on Computational Sciences and Applications*, Vol. 4, No. 1, pp. 103-112, 2014.

[22] M. Alazab et al., "A Hybrid Wrapper-Filter Approach for Malware Detection", *Journal of Networks*, Vol. 9, No. 11, pp. 2878-2891, 2014.

[23] S. Singla, E. Gandotra, D. Bansal and S. Sofat, "A Novel Approach to Malware Detection using Static Classification", International Journal of Computer Science and Information Security, Vol. 13, No. 3, pp. 1-5, 2015.

[24] S. Chaumette, O. Ly and R. Tabary, "Automated Extraction of Polymorphic Virus Signatures using Abstract Interpretation", *Proceedings of 5th International Conference on Network and System Security*, pp. 41-48, 2011.

[25] A. Verma, M. Rao, A. Gupta, W. Jeberson and V. Singh, "A Literature Review on Malware and Its Analysis", *International Journal of Current Research and Review*, Vol. 5, No. 16, pp. 71-82, 2013.

[26] S. Ranveer and S. Hiray, "Comparative Analysis of Feature Extraction Methods of Malware Detection", *International Journal of Computer Applications*, Vol. 120, No. 5, pp. 1-7, 2015.

[27] L. Wang, Z. Li, Y. Chen, Z.J. Fu and X. Li, "Thwarting Zero-Day Polymorphic Worms with Network-Level Length-based Signature Generation", *IEEE/ACM Transactions on Networking*, Vol. 18, No. 1, pp. 53-66, 2010.

[28] M.A.I. Almarshad, M.M.Z.E. Mohammed and A.S.K. Pathan, "Detecting Zero-Day Polymorphic Worms with Jaccard Similarity Algorithm", *International Journal of Communication Networks and Information Security*, Vol. 8, No. 3, pp. 203-214, 2016.

[29] Y. Prayudi and S. Yusirwan, "The Recognize of Malware Characteristics Through Static and Dynamic Analysis Approach as an Effort to Prevent Cybercrime Activities", *Journal of Theoretical and Applied Information Technology*, Vol. 77, No. 3, pp. 438-445, 2015.

[30] M. Vasilescu, L. Gheorghe and N. Tapus, "Practical Malware Analysis based on Sandboxing", *Proceedings of 13th Edition: Networking in Education and Research*, pp. 1-6, 2014.

[31] U. Baldangombo, N. Jambaljav and S.J. Horng, "A Static Malware Detection System using Data Mining Methods", *International Journal of Artificial Intelligence and Applications*, Vol. 4, No. 4, pp. 113-119, 2013.

[32] M. Sikorski and A. Honig, "*Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*", 1st Edition, No Starch Press, 2012.

[33] J.U. Joo, I. Shin and M. Kim, "Efficient Methods to Trigger Adversarial Behaviors from Malware during Virtual Execution in Sandbox", *International Journal of Artificial Intelligence and Applications*, Vol. 9, No. 1, pp. 369-376, 2015.

[34] S. Gadhiya and K. Bhavsar, "Techniques for Malware Analysis", *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 3, No. 4, pp. 2277-2281, 2013.

[35] J. Landage and M. Wankhade, "Malware and Malware Detection Techniques: A Survey", *International Journal of Engineering Research*, Vol. 2, No. 12, pp. 61-68, 2013.

[36] S.K. Pandey and B.M. Mehtre, "Performance of Malware Detection Tools: A Comparison", *Proceedings of IEEE International Conference on Advanced Communication, Control and Computing Technologies*, pp. 1811-1817, 2015.

[37] S. Yusirwan, Y. Prayudi and I. Riadi, "Implementation of Malware Analysis using Static and Dynamic Analysis Method", *International Journal of Computer Applications*, Vol. 117, No. 6, pp. 11-15, 2015.

[38] S. Hong and S. Lee, "New Malware Analysis Method on Digital Forensics", *Indian Journal of Science and Technology*, Vol. 8, No. 17, pp. 1-6, 2015.

[39] A. Kumar, K.S. Kuppusamy and G. Aghila, "A Learning Model to Detect Maliciousness of Portable Executable using Integrated Feature Set", *Journal of King Saud University-Computer and Information Sciences*, 2017.

[40] M.Z.A. Bhuiyan, J. Wu, G.M. Weiss, T. Hayajneh, T. Wang and G. Wang, "Event Detection through Differential Pattern Mining in Cyber-Physical Systems", *IEEE Transactions on Big Data*, 2017.

[41] J. Drew, T. Moore and M. Hahsler, "Polymorphic Malware Detection using Sequence Classification Methods", *Proceedings of IEEE Security and Privacy Workshops*, pp. 81-87, 2016.

[42] J. Drew, M. Hahsler and T. Moore, "Polymorphic Malware Detection using Sequence Classification Methods and Ensembles", *EURASIP Journal on Information Security*, Vol. 2017, No. 1, pp. 1-2, 2017.

[43] P. Sharma, S. Kaur and J. Arora, "An Advanced Approach to Polymorphic/Metamorphic Malware Detection using Hybrid Clustering Approach", *International Research Journal of Engineering and Technology*, Vol. 3, No. 6, pp. 2229-2232, 2016.