I.J. Information Technology and Computer Science, **2016, 4, 63-73**
Published Online April 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2016.04.08

Hybrid Real-time Zero-day Malware Analysis and Reporting System

Ratinder Kaur and Maninder Singh
Computer Science and Engineering Department, Thapar University, Patiala-147004, India
E-mail: ratinder@thapar.edu, msingh@thapar.edu

Abstract—To understand completely the malicious intents of a zero-day malware there is really no automated way. There is no single best approach for malware analysis so it demands to combine existing static, dynamic and manual malware analysis techniques in a single unit. In this paper a hybrid real-time analysis and reporting system is presented. The proposed system integrates various malware analysis tools and utilities in a component-based architecture. The system automatically provides detail result about zero-day malware's behavior. The ultimate goal of this analysis and reporting is to gain a quick and brief understanding of the malicious activity performed by a zero-day malware while minimizing the time frame between the detection of zero-day attack and generation of a security solution. The results are paramount valuable for a malware analyst to perform zero-day malware detection and containment.

Index Terms—Zero-day Attacks, Unknown Attacks, Static Analysis, Dynamic Analysis, Malware Reporting.

I. Introduction

Zero day attacks are reality and their number reported each year increases immensely. In recent years, zero-day attacks have been dominating the headlines for political and monetary gains. They are a potent weapon in the hands of attackers and are being used as essential success vectors in various sophisticated and targeted attacks. These secret weapons give attackers a crucial advantage over their targets to break into traditional security products that identify only known, confirmed threats. Attackers always deploy the latest technology and constantly change techniques to infiltrate systems. The zero-day attacks are among the top security concerns that the modern enterprises face today. People talked about zero-day attacks few years back, but today every industry faces it. Reports and news on the Internet security shows an alarming increase of such attacks against both corporate and home user systems [1].

McAfee Labs [2], Panda Labs [3] reported that the sheer number of unique malware samples grows exponentially every year. Attackers use automated tool kits to generate several thousand malware variants at once with armoring techniques like run-time obfuscation, polymorphism and packers. It is estimated by security experts that more than 70,000 new instances of malware are released each day [4]. Thousands of new malwares are emerging and the existing malwares are evolving in their structure every single day to achieve stealth with respect to standard intrusion detection and malware analysis techniques. In the present scenario the existing detection and analysis methods are inefficient to deal with the exponential growth of zero-day malware arising from innumerable automated obfuscations.

To defend against zero-day malware there has been a shift from signature-based [5-8, 38] to anomaly-based detection [9] and behavioral-based detection [10-16, 39]. Various behavior-based detection techniques have been proposed that understands the behavior of zero-day malware through dynamic execution [10, 11]. Behavior based techniques look for the essential characteristics (indicators) of malware which do not require the examination of payload byte patterns. They focus on the actual dynamics of the malware execution to detect them. They monitor the behavior of malicious software in a controlled environment, no matter what, a piece of malware will behave badly while running. This is an effective way to detect zero-day malware without waiting for them to do any harm. For behavior-based detection there is a need to monitor the events that characterize the execution of the malicious program. The most promising and effective technique to characterize the behavior of a program is to monitor the system call functions. System calls provide an intrinsic abstraction of a set of actions executed by malware. All variants of one malware exhibit similar behavior and samples with same functionality may also have similar behavior. Existing system call behavioral models [17-19] are derived from the results of malware analysis.

Malware analysis is the art of dissecting malware to understand how it works, how to identify it, and how to defeat or eliminate it [20]. It is a critical task for responding to computer or network security incidents as it allows to better assess the nature of a security incident and may even help to prevent further infections. Therefore, malware analysis is a necessary to develop an effective detection technique. To turn malicious programs inside out and to understand their inner workings a core set of tools and techniques is required for analyzing. Analyzing malware is a time-consuming task and it usually involves notable manual effort which by itself requires significant expertise to be carried out. The traditional approach of malware analysis requires lots of manual effort. It involves: (1) allocating physical or virtual systems for the analysis, (2) isolating analysis

systems from the production environment, (3) installing several open-source or commercial behavioral analysis tools and code-analysis tools, and (4) utilizing online analysis tools. It is a tedious time intensive process which increases observation duration. This traditional approach is becoming powerless as the number of malware samples are constantly evolving and increasing. The manual analysis work increases included delay between the detection of zero-day malware and its containment.

In this paper a hybrid real-time zero-day malware analysis and reporting system is proposed. The proposed system integrates existing malware analysis tools and techniques in a component based architecture to work as a single unit. It combines the advantages of static, dynamic, and manual analysis to generate a comprehensive report on zero-day malware behavior. This paper makes the following contributions:

- The proposed system addresses the research problems with existing approaches in zero-day malware analysis automatically with minimal manual intervention. It aims at integrating traditional steps involved in malware analysis.
- The proposed system integrates the functionalities of static, dynamic, and manual analysis to generate a comprehensive report on zero-day malware behavior. Hence, decreasing actual analysis time.
- The proposed system is able to analyze malware that employ anti-analysis techniques to detect virtual or emulated environment. The system uses real host with Operating System restore backup for analysis hence, reducing virtualization and emulation overheads.

The remainder of the paper is organized as follows. In Section 2, related work is summarized. In Section 3, the detailed working of the proposed system is presented. Finally Section 4, describes the results and the paper is concluded in Section 5.

II. Related Work

There are basic two approaches for malware analysis, which security professionals perform: Static (Code) Analysis and Dynamic (Behavioral) Analysis. Although both types accomplish the same goal of explaining how malware works, the tools, time and skills required to perform the analysis are very different.

The static analysis allows to learn malware's capabilities by examining the code from which the program was comprised. While performing static analysis anti-virus software is run to confirm maliciousness, hashes are used to identify malware, strings are searched, functions, headers and scripts are analyzed. Static analysis is mostly conducted manually and can be applied on different representations of a program. If the source code is available, information such as variables, data structures, used functions and call graphs can be extracted. Static analysis is also used on the binary representation of a program. Static analysis is tricky and time-consuming, because source code of malware is not always available. Instead, the complied executable's functionality is examined at the assembly level using a disassembler such as IDA Pro [21], which converts the instructions from their binary form into the human-readable assembly form.

Various static malware analysis methods have been proposed [22-24]. Static analysis offers a significant improvement in malware detection accuracy while compared to traditional pattern matching. But its main weakness lies in the difficulty to handle obfuscated and self-modifying code [25]. Eureka [26] provides a malware de-obfuscation framework, to assist in static analysis. It uses a novel binary unpacking strategy based on statistical bigram analysis and coarse-grained execution tracing. MaTR [27] combines machine learning algorithm with static heuristic features for unknown malware detection. A program analysis tool [28] is proposed to automatically derive data invariants from source code, using static analysis. The tool applies compiler technology to analyze the control and data flows (e.g., assignments, function calls, and conditional statements) of a target program and hypothesizes likely invariants (e.g., constant, membership, bounds, and non-zero). API-CFG [29] extracts control flow graphs from programs and combines it with extracted API calls to have more information about PE files.

During dynamic analysis it is examined how the malware behaves and interacts with its environment when executed. In dynamic analysis the malware is executed on an isolated or virtual system, its interaction with overall system including file system, registry, system processes and network is observed [37]. Sometimes, it is required to interact with the malware to discover its additional characteristics and for this debuggers are used to examine the internal state of a running malware. Generally, there are two main approaches for dynamic malware analysis. (1) Analyzing the difference between defined states: A given malware is executed for a certain period of time and afterwards the modifications made to the system are analyzed by comparison to the initial system state. In this approach, comparison report states behavior of malware. (2) Observing runtime-behavior: In this approach, malicious activities launched by the malicious application are monitored during runtime using a specialized tool.

Various automated dynamic malware analysis tools and frameworks have been proposed. These tools execute an unknown malware in an instrumented environment and monitor its execution. The analysis reports generated by these tools provide insights about the behavior of running malware. Anubis stands for Analyzing Unknown Binaries focuses on automated dynamic malware analysis. It evolved from TTAnalyze [30] and executes the sample under analysis in an emulated environment consisting of a Windows XP running as the guest in a modified version of Qemu [31]. The analysis is performed by monitoring the invocation of Windows API functions, as well as system service calls to the Windows Native API. Cuckoo Sandbox [32] is an open-source tool for dynamic malware analysis that uses the technique of API-hooking. The actual instrumentation of the running processes is

done by injecting a dynamic linked library (DLL) that hooks Windows API functions and logs their parameters when called. This DLL also randomize the instructions written to the target function in order to evade anti-analysis techniques used by modern malwares. CWSandbox [33] uses API-hooking and code injection technique to analyze malware dynamically. It executes the malware either natively or in a virtual Windows environment. The sandbox injects a monitoring DLL in the malware process, which implements API hook functions to trace relevant system calls. Norman Sandbox [34] emulates whole computer and a network connected to it. Norman Sandbox executes the sample in a tightly-controlled environment that simulates a Windows OS, attached local area network (LAN) and some Internet connectivity. Norman Sandbox focuses on the detection of worms that spread via email or P2P networks, as well as viruses that try to replicate over network shares. Norman Sandbox also uses function call hooking and parameter monitoring techniques to detect malware. Joe Sandbox previously known as JoeBox [35] is specifically designed to run on real hardware. Joe Sandbox uses client server model, where a single controller instance coordinates multiple clients that are responsible for performing the malware analysis and all analysis data is collected by the controlling machine.

III. Proposed System

The proposed system is a collection of well-known malware analysis tools and techniques in a component-based architecture, where any tool can be replaced in the future. The tools have been modified and integrated into the system to behave as a single unit. The integrated tools and techniques work together automatically and to provide detailed and efficient result in zero-day malware behavior. Fig. 1, depicts the basic components of the

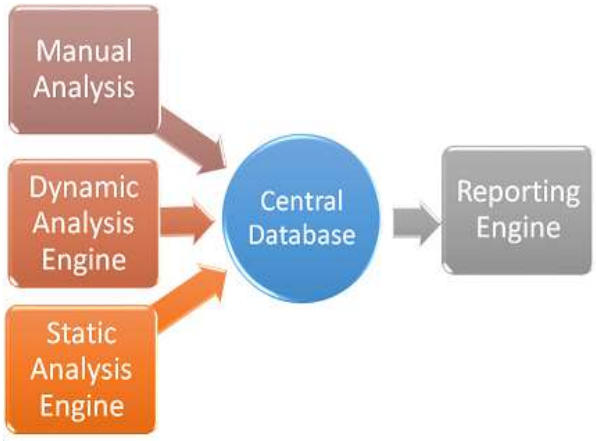

Fig.1. Basic System Components

proposed system. Static Analysis Engine (SAE) obtains basic information of the zero-day malware and stores a structural profile in central database. Dynamic Analysis Engine (DAE) records all the execution activities and stores a behavioral profile in the database. If the malware analyst requires more insight about malware behavior then manual analysis can also be done and results can be updated. These analysis results are accessed by the reporting engine that generates zero-day malware analysis report in a HTML or PDF format. Here the main focus is on analyzing malicious Windows PE files.

3.1 Static Analysis Engine

SAE comprises of static analysis functions, running parallel in the background in the analysis server as in Fig. 2. The zero-day malware is checked for static properties and findings are reported. SAE reports about antivirus scanning, obfuscation, PE structure, hashes and strings. All these static analysis functions are included by integrating popular static analysis tools/utilities. SAE is completely modular and this makes it flexible and extensible. SAE has a main python script *static.py*, which starts each functionality and extracts its output to save in database. With the preliminary static analysis it is possible to extract valuable information that will shape the profile of the malware. The integrated static functions are:

Antivirus Scanning:

As various antivirus programs uses different signatures and heuristics, it is useful to scan suspected malware against different antivirus programs. Therefore, VirusTotal [36] is used in the first analyzing step to check which antivirus programs have already identified the malware under question. VirusTotal provides free checking of files for malware. It uses more than 50 different antivirus products and scan engines. VirusTotal generates a report that provides information about the suspected malware. It report analysis details like malware name, file size, hash, if available, additional behavioral information about the malware and the detection rate (total number of antivirus products that marked the file as malicious divided by total number of antivirus products). The malware is scanned by the VirusTotal to check whether same binary (pe_file) has been earlier identified by other antivirus program or not. For this VirusTotal Public API v2.0 is used.

Sending file:
```
host = "www.virustotal.com"
selector = "https://www.virustotal.com/vtapi/v2/file/scan"
fields = [("apikey", "xyz")]
file_to_send = open("pe_file", "rb").read()
files = [("file", "pe_file", file_to_send)]
json = postfile.post_multipart(host, selector, fields, files)
```

Retrieving scan report:
```
url = "https://www.virustotal.com/vtapi/v2/file/report"
parameters = {"resource": "md5 of pe_file", "apikey": "xyz"}
data = urllib.urlencode(parameters)
req = urllib2.Request(url, data)
response = urllib2.urlopen(req)
json = response.read()
```

The VirusTotal APIs uses HTTP POST request with JSON object response format for sending and retrieving scan reports respectively. The request message contains host details, file information and apikey (for accessing

public APIs). For each sent HTTP POST request the JSON response object contains a parameter known as *response_code*, which determines the response result. If the item searched is not present in VirusTotal's dataset, *response_code* will be 0. If the requested item is still queued for analysis it will be -2. If the item is present and it could be retrieved it will be 1. Following code snippet depicts how binary is sent and response is retrieved.

If at first step *response_code* is 0, then the binary is processed further for analysis by SAE.

Obfuscation:

Malwares often use obfuscation techniques to evade detection systems. One such popular obfuscation technique is packing. To detect the type of packer employed PEid has been utilized. To integrate this feature, PEid database, peidDB.txt, is accessed which contains 1832 packer signatures. Once the database is loaded, the malware is read for matching packer signature. An option is also provided to add more signatures later in the database file or to load an alternative database for aggregating more signatures. The database file has packer name as the section name and two keys: the signature key containing the byte pattern and the ep_only key. The ep_only property can be true or false. This property specifies if the signature has to be found at the PE file's entry point (true) or can be found anywhere (false). The malware is scanned to find the matching packer signature which is then updated in the central database.

```
pe = pefile.PE(pe_file)
signatures = peutils.SignatureDatabase('peidDB.txt')
matches = signatures.match_all(pe, ep_only = True)
update "matches" in database
```

PE Header Information:

Any executable file includes a header to describe its structure like, the base address of code section, data section, list of functions imported, exported, etc. To execute the file, the Operating System simply reads the header first and loads the binary data from the file to code/data segments of the address space for the corresponding process. During dynamic linking the Operating System relies on file's import table to determine the entry addresses of the system functions. Most executable files on Windows follows the following structure: DOS Header (64 bytes), PE Header, sections (code and data). DOS Header starts with magic number 4D 5A 50 00, and the last 4 bytes is the location of PE header in the binary file. The PE header contains significantly more information and is more interesting. At run time, Windows loader loads the PE header into a process's address space. PE header consists of three parts: (1) a 4-byte magic code, (2) a 20-byte file header and its data type is IMAGE_FILE_HEADER, and (3) a 224-byte optional header (type: IMAGE_OPTIONAL_HEADER32). The optional header itself has two parts: the first 96 bytes contain information such as major operating systems, entry point, etc. The second part is a data directory of 128 bytes. It consists of 16 entries, and each entry has 8 bytes (address, size). The PE header contains useful information for the malware analyst and the important fields that can be obtained from a PE header are:

- Imports: Functions from other libraries that are used by the malware.
- Exports: Functions in the malware that are meant to be called by other programs or libraries.
- Time Date Stamp: Time when the program was compiled.
- Sections: Names of sections in the file and their sizes on disk and in memory.
- Subsystem: Indicates whether the program is a command-line or GUI application.
- Resources: Icons, menus, and other information included in the file.

To extract this valuable information the SAE uses a Python PE parsing module, Pefile 1.2.10-139, to inspect PE header, to retrieve all the sections, imports, exports, resources, their information and data. The output is get in the desired format and stored in the central database.

```
# Attributes
Image Base: hex(pe.OPTIONAL_HEADER.ImageBase)
Address Of Entry Point:
hex(pe.OPTIONAL_HEADER.AddressOfEntryPoint)
Required CPU type: pefile.MACHINE_TYPE[machine]
dll = pe.FILE_HEADER.IMAGE_FILE_DLL
Subsystem:
pefile.SUBSYSTEM_TYPE[pe.OPTIONAL_HEADER.Subsyste
m]
Compile Time:
datetime.datetime.fromtimestamp(pe.FILE_HEADER.TimeDateSt
amp)
Number of RVA and Sizes:
pe.OPTIONAL_HEADER.NumberOfRvaAndSizes

# Sections
Number of Sections: pe.FILE_HEADER.NumberOfSections
for section in pe.sections:
section.Name, hex(section.VirtualAddress),
hex(section.Misc_VirtualSize),\
section.SizeOfRawData, E(section.data)

#Resources
For res in pe.DIRECTORY_ENTRY_RESOURCE.entries
update "res.name, res.data.struct.OffsetToData,
res.data.struct.Size, res.filetype, res.data.lang" in database

# Imports
for entry in pe.DIRECTORY_ENTRY_IMPORT:
update "entry.dll" in database
for imp in entry.imports:
update "hex(imp.address), imp.name" in database

# Exports
for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
update hex(pe.OPTIONAL_HEADER.ImageBase + exp.address),
exp.name, exp.ordinal
```

Hashing:

The main purpose of using this feature is to generate various hashes for the binary. These hashes provides a unique fingerprint for the malware. The SAE generates various hashes like MD5, SHA-1 and SHA-256 for the malware. SAE also returns a ten digit representation of

the size of file processed. Therefore, along with hash value the file size is extracted as well and saved in database. This functionality is implemented by the system. For this *hashlib* module is used which implements a common interface to many different secure hash and message digest algorithms.

```
fileStr= open('pe_file','rb').read()
hashlib.md5(fileStr).hexdigest()
hashlib.sha1(fileStr).hexdigest()
hashlib.sha256(fileStr).hexdigest()
update in database
```

Strings:

A malware program contains strings if it has to print a message, connect to a URL, or has to copy a file to a specific location. Searching these strings can help to get hints about the program functionality. Like, the legitimate programs always include many embedded strings but an obfuscated or packed malicious program contains very few strings. So, if few embedded strings are returned, either make sense or not, then the tested binary is likely to be malicious. SAE examines ASCII and Unicode strings in binary data. All the printable strings from the binary file are saved in database and reported. This is also system implemented using *string* python module.

```
fileStr=open(pe_file, 'rb').read()
if fileStr in string.printable:
result +=  fileStr
update database
```

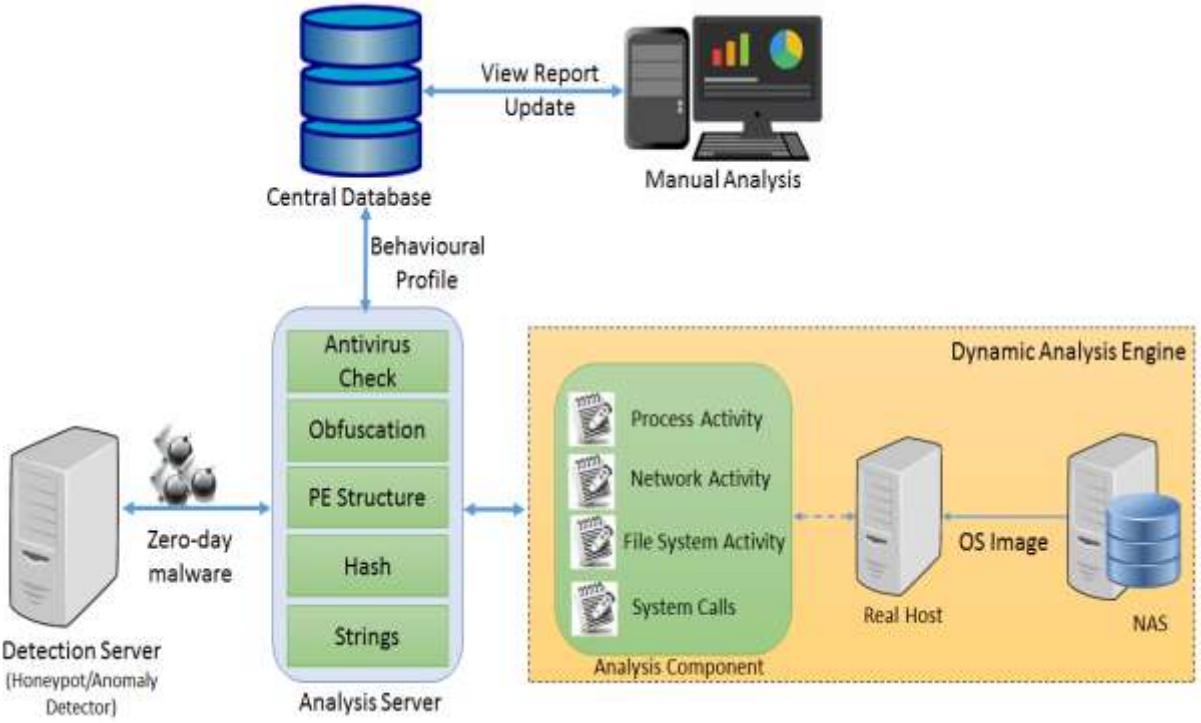

Fig.2. Zero-day Malware Analysis and Reporting System

3.2 Dynamic Analysis Engine

After reporting static properties the binary is passed to DAE for dynamic analysis since static analysis is not foolproof. DAE focuses on "behavioral analysis", by executing and monitoring the malware. This helps to understand the nature and the purpose of the zero-day malware and reveals which files are read or accessed and which operations has been carried out. DAE (Figure 2) comprises of an analysis component, a real host and a network attached storage (NAS). It is possible that malware authors design their malware to check execution environment by employing anti-analysis techniques. If the execution environment is detected, the malware can either stop running or raise an exception or loop for a long time, thus evading its detection. The proposed system addresses this problem by running the malware on a real host isolated from the production network. The real host has an advantage of real Operating System services and applications, which helps to provide more authentic behavioral information about the unknown malware. Zero-day malware is executed on real hardware without relying on any virtualization or emulation techniques. The real host is a Windows 7 machine which runs integrated analysis tools and utilities to track various system activities performed by the malware. After each execution on real host, the original system image is restored from NAS for next execution. The analysis component logs high level information of the malicious actions regarding file system activity, registry activity, process activity and network activity. These logs are then sent to analysis server and uploaded to central database. Following dynamic analysis features are incorporated in the analysis component for automatic monitoring. They use windows utilities combined to run in one batch file.

Process Activity:

This functionality provides insight into processes currently running on a system when malware is executing. When the malware is executed, the active processes are monitored to identify loaded DLLs, run-time DLLs, memory statistics, threads and open handles. *PsList* utility is used to dump statistics like memory usage and thread detail for the running malware. *ListDlls* and *Handle* utilities return DLLs loaded and open Operating System resource handles (such as a file, directory or registry key) respectively.

PsList = <*Name, Pid, Pri, Thd, Hnd, Priv, CPU_Time, Elapsed_Time*>
PsList –m = <*Virtual_Mem, Working_Set, Priv_Virt_Mem, Priv_Virt_Mem_Peak, Page_Faults, Non_Paged_Pool, Paged_Pool* >
ListDlls = <*Module_Name, Version, Base_Address, Size*>
Handle = <*Handle_Value, Object_Type, Object_Name*>

Network Activity:

It is important to keep a check on network connections and this functionality provides information about active connections established by the running malware. This functionality retrieves network information from the system like network connections (both incoming and outgoing), number of bytes transferred and network protocol statistics. It also records network traffic for malicious communication attempts, such as DNS resolution requests, bot traffic, or downloads. *Netstat* and *Tshark* (the command line version of Wireshark) utilities are integrated. Network logs and Pcaps are also captured.

Netstat = <*Proto, Local_Address, Foreign_Address, State*>
Tshark -i, -p -a "filesize" -w "log.pcap"= <*Pcap_Logs*>

System Calls:

System calls provide useful information about process behavior. So, to intercept and record the system calls which are called by a process and the signals which are received by a process, *strace* utility is used. It monitors interactions between processes and the Operating System kernel, which include system calls, signal deliveries, and changes of process state.

Strace –p "pid" = <*System_Call, Args, Return_Value, Exe_Time*>

File System Activity:

This functionality monitors real-time file system and registry activity. It returns list of added, deleted and modified files and registry keys. *Procmon* is used to log file system and registry changes.

```
set PM=C:\sysint\procmon.exe
start %PM% /quiet /minimized /backingfile  logs.pml
%PM% /waitforidle
pe.exe
%PM% /terminate
```

3.3 Manual Analysis

It is an indispensable step in analyzing zero-day attacks as both static analysis and dynamic analysis have their own limitations. However, the information collected from both static analysis and dynamic analysis will be useful for a human analyst while dissecting a zero-day binary. But still, if some part of analysis is left in SAE and DAE then that can be manually performed by an expert. For this the binary is run in a debugger, *OllyDbg*, to animate instructions in a slow and controlled fashion. To do so, the Ctrl+F8 (ANIMATE OVER) is used to stepover until an address is arrived, which is the call to the main function. Next, the Crtl+F7 (ANIMATE INTO) is used to step-into the call to the main function. This is continued to step forward using F7 and F8 while noting the behavior of the sample. To evade anti-debugging techniques of malicious binary, anti-anti-debugger plugin (aadp) for *OllyDbg* has been used. The aadp plugin avoids anti-debugging techniques like anti-debugging APIs or flags. In debugger a running program can be resumed in three different ways:

- breakpoint: stops a program whenever a particular point in the program is reached.
- watchpoint: stops a program whenever the value of a variable or expression changes.
- catchpoint: stops a program whenever a particular event occurs, analyze CPU environment (memory, registers).

3.4 Reporting Engine

The proposed system generates a zero-day malware analysis report in HTML and PDF format using JasperReports. Firstly, a report template (.jrxml file) is created. This template file is then complied to get a Jasper object which is further processed using Java utility to populate the data. Finally, Jasper print file is exported to HTML and PDF format. The report is generated from static and dynamic analysis data uploaded by the analysis server in central database. Additionally, manual analysis findings are uploaded directly in form of notes by the analyst. Fig. 3, shows reporting attributes captured. All the attributes are captured during analysis phase and are used to generate analysis report which covers following areas:

- Analysis Summary: Key outcome from the analysis report regarding the malware's nature, origin, capabilities, relevant characteristics, indicators of compromise, follow-up actions and lessons learned.
- Identification: The file type, its size, hashes (such as MD5, SHA1, and SHA256), file name, anti-virus detection potential.
- Characteristics: Capability to infect files, self-preserve, spread, data leakage, communicate with the attacker, etc.
- Dependencies: System resources (like files,

network, and memory) related to the malware's functionality, initialization files, DLLs, executables, URLs, and scripts.

- Supporting Artifacts: Logs, pcaps, dumps, string extracts, function listings, figures and other relevant system and network statistics.
- Manual Analysis Findings: Overview of the manually done static and dynamic code analysis observations.

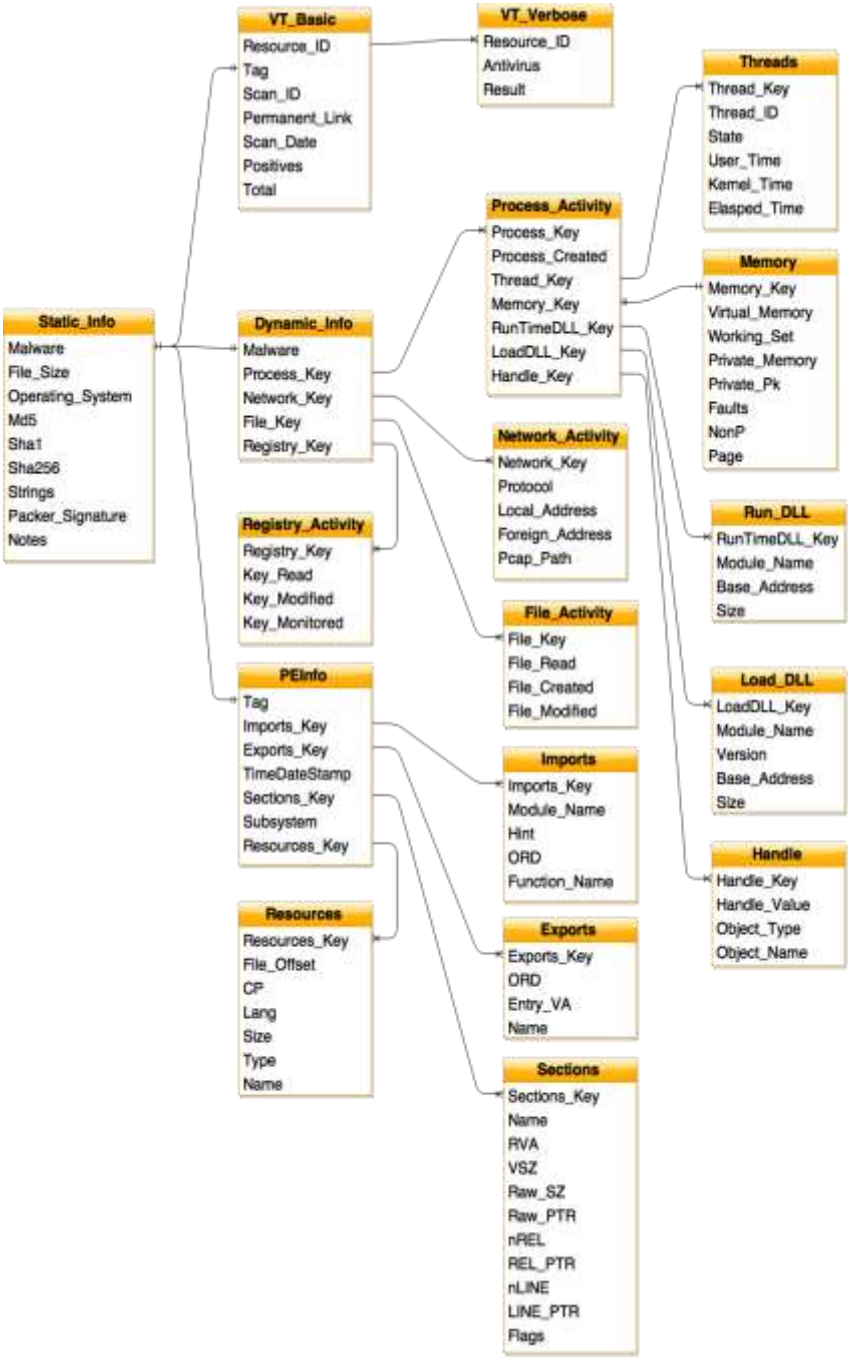

Fig.3. Reporting Attributes

Algorithm 1 represents the big picture, the bird's-eye view of the total zero-day malware analysis and reporting system's working.

Algorithm 1: Zero-day Analysis & Reporting

```
1: procedure Analysis()
2:      List malware_list = getMalware();
3:      for malware in malware_list do
4: SAE(malware);
5: DAE(malware);
6:      end for
7: end procedure
8: function List getMalware()
9:    Read pe_file from /usr/home/PE_files/.
10: return pe_file;
11: end function
12: function SAE(pe_file)
13:      invoke uploadVirusTotal(pe_file);
14:      repeat
15:           response = getVirusTotalResponse();
16:      until response==null
17:      if (detectionRatio != 0) then
18:           Upload VirusTotal result in database.
19:           BREAK;
20:      else Continue;
21:      end if
22:      packer = obfuscation(pe_file);
23:      Upload packer information
24:      header= PEstructure(pe_file);
25:      Upload header
26:      hash= hash(pe_file);
27       Upload md5sum, sha
28:      response[] = strings(pe_file);
29:      Upload list of embedded strings in database
30: end function
31: function DAE(pe_file)
32:      process_activity [0] = PSList(pe_file);
33:      process_activity [1] = ListDlls(pe_file);
34:      process_activity [2] = Handle(pe_file);
35:      Upload database process_activity
36:      network_activity [0] = netstat();
37:      network_activity [1]= tshark();
38:      Upload database network_activity
39:      system_activity [] = strace(pe_file);
40:      Upload database system_activity
41:      file_activity [] = filemon(pe_file);
42:      registry_activity []= regmon(pe_file);
43:      Upload database file_activity and registry_activity
44: end function
45: function Debugger(pe_file)
46:      Manually add and upload notes
47: end function
48: function Reporting(pe_file)
49:      Create .jrxml file using iReport
50:      Complie .jrxml to .Jasper
51:      Fill .Jasper with data from central database
52:      Export the report into HTML or PDF
53: end function
```

IV. RESULTS

To evaluate the proposed system a prototype was implemented using the Oracle Java6 SDK, Eclipse IDE, Python and MySql database. Various off-the-shelf solutions have been employed wherever possible in an attempt to allow existing tools and utilities to be integrated into the system. For that many existing tools and utilities were modified and incorporated in the system to work as a single unit. The zero-day malware from detection server (it can be honeypot or anomaly detector) is forwarded to analysis server for static and dynamic analysis. A real host machine is connected directly to the analysis server for executing the malware in controlled environment. After every execution in the real host, the Operating System image is updated from NAS. All the analysis results are stored in the central database. Reporting server fetches the analysis result from the database and represent it in the form of HTML or PDF format reports. A manual analysis machine is also attached to the database from where the malware analyst can view the reports and at the same time can do debugging of the malware (if required) and update results.

Fig.4. Basic Static Information

Some of the snapshots from the generated report are shown. They present the behavior of an email worm captured from detection server during validation. The worm code was tweaked and packed to act as unknown for our system. Fig. 4, represents general static information captured for the email worm sample. Static information like file name, file size, its MD5 sum, packer with which it was encrypted and file header information-header sections, imports, exports, resources. This static information helps to structure the profile of a malware.

Fig. 5, shows process activities performed by the malware while it was executed. Process activity shows load dlls (dynamic link libraries), run-time dlls, mutexes created by the worm sample or any exceptions thrown.

Fig.5. Process Activity

Fig. 6, shows network communication of the worm sample. It does DNS queries and uses this information to send email over SMTP connections. The worm sends DNS queries to xyz.mail.yahoo.com domains to get their respective IP addresses. It then sends email over SMTP to those IP addresses with a malicious attachment.

Unknown traffic over UDP by the worm is also recorded A normal UDP connection is established and terminated after sending and receiving some bytes of data.

DNS Queries	
e.mx.mail.yahoo.com	QueryType: DNS_TYPE_A, Result: 216.39.53.1
g.mx.mail.yahoo.com	QueryType: DNS_TYPE_A, Result: 206.190.53.191
a.mx.mail.yahoo.com	QueryType: DNS_TYPE_A, Result: 67.195.168.31
f.mx.mail.yahoo.com	QueryType: DNS_TYPE_A, Result: 68.142.202.247
SMTP Connections	
Server:1038 to 206.190.53.191:25	Email with attached file: "old_photos.scr"
Server:1037 to 67.195.168.31:25	Email with attached file: "myaunt_unfolds.com"
Server:1040 to 68.142.202.247:25	Email with attached file: "ranking_birth.zip"
Server:1039 to 216.39.53.1:25	Email with attached file: "warez.pif"
Unknown UDP Traffic	
Server:1033 to 192.168.0.1:53	State: Normal establishment and termination, Outbound Bytes: 28, Inbound Bytes: 511
Server:1032 to 192.168.0.1:53	State: Normal establishment and termination, Outbound Bytes: 28, Inbound Bytes: 511
Server:1034 to 192.168.0.1:53	State: Normal establishment and termination, Outbound Bytes: 28, Inbound Bytes: 511
Server:1035 to 192.168.0.1:53	State: Normal establishment and termination, Outbound Bytes: 28, Inbound Bytes: 511

Fig.6. Network Activity

Created:	C:\WINDOWS\ranking_birth.zip C:\WINDOWS\winlogon.exe
Read:	C:\WINDOWS\ranking_birth.zip PIPE\lsarpc c:\autoexec.bat c:\documents and settings\administrator\application data\microsoft\internet explorer\brndlog.txt c:\documents and settings\administrator\local settings\application data\iconcache.db
Modified:	C:\WINDOWS\ranking_birth.zip C:\WINDOWS\winlogon.exe PIPE\lsarpc \Device\Afd\Endpoint \Device\NetBT_Tcpip_{82851064-BBF5-4528-B62B-E6D62A782874} \Device\RasAcd \Device\Tcp

Fig.7. File System Activity

Fig.7, shows file system access by the worm sample while it was running. Files created, read and modified are listed in the report.

Fig.8., shows system registry access by the worm sample during execution. Registry keys modified, read and monitored are listed in the report.

Modified:	
HKLM\SOFTWARE\Microsoft\Windows\CurrentVers	Name: ICQ.Net, New Value: C:\WINDOWS\winlogon.exe -stealth
HKLM\SYSTEM\CURRENTCONTROLSET\HARDWARE PROFILES\CURRENT\Software\Microsoft\windows\ CurrentVersion\Internet Settings	Name: ProxyEnable, New Value: 0
HKLM\Software\Microsoft\Windows\CurrentVersio n\Explorer\Shell Folders	Name: Common AppData, New Value: C:\Documents and Settings\All Users\Application Data
HKU\S-1-5-21-1229272821-1004336348-527237240-1003\Software\Microsoft\Windows\CurrentVersio n\Explorer\Shell Folders	Name: AppData, New Value: C:\Documents and Settings\user\Application Data
HKU\S-1-5-21-1229272821-1004336348-527237240-1003\Software\Microsoft\Windows\CurrentVersio n\Explorer\Shell Folders	Name: Cache, New Value: C:\Documents and Settings\user\Local Settings\Temporary Internet Files ... [For more click here]
Read:	
HKLM\SOFTWARE\Microsoft\Windows\CurrentVers ion\Internet Settings	Name: UrlEncoding, Value: 0x00A2D0, Times: 2
HKLM\SYSTEM\CONTROLSET001\SERVICES\NETBT\ PARAMETERS	Name: DhcpNodeType, Value: 4, Times: 8
HKLM\SYSTEM\CONTROLSET001\SERVICES\TCPIP\P ARAMETERS	Name: Hostname, Value: user, Times: 16
HKLM\SYSTEM\Setup	Name: SystemSetupInProgress, Value: 0, Times: 1
HKLM\Software\Microsoft\Tracing\RASAPI32	Name: FileDirectory, Value: %windir%\tracing, Times: 4 ... [For more click here]
Monitored:	
HKLM\Software\Microsoft\Tracing\RASAPI32	NotifyFilter: Attributes Change,Value Change,Security Descriptor
HKLM\System\CurrentControlSet\Services\WinSoc k2\Parameters\Namespace_Catalog	NotifyFilter: Key Change
HKLM\System\CurrentControlSet\Services\WinSoc k2\Parameters\Protocol_Catalog	NotifyFilter: Key Change

Fig.8. Registry Activity

Graph in Fig. 9, depicts high level file system behavior. Worm sample does maximum registry read activity which is 77% of overall file system activity.

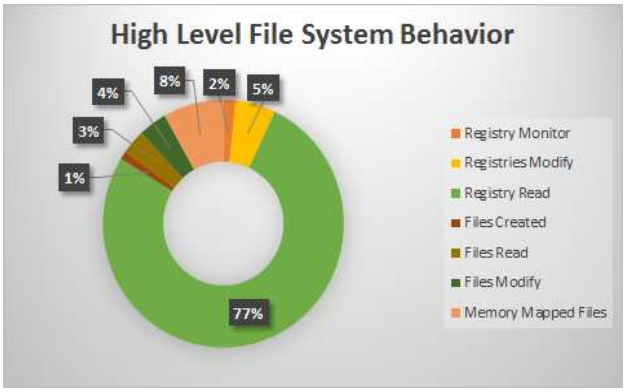

Fig.9. File System Access

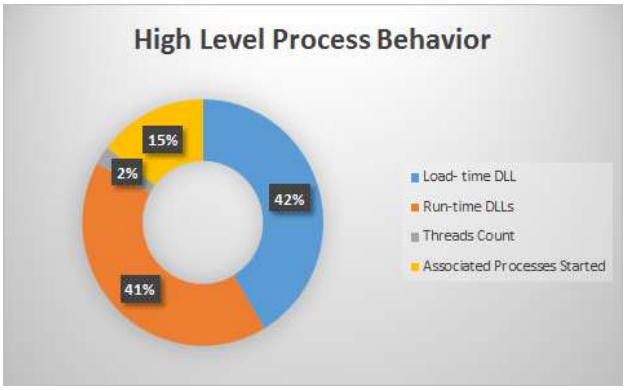

Fig.10. Process Behavior

Graph in Fig. 10, depicts high level process behavior. Worm sample further started 2 threads and 15 other processes for its working. Rest shows percentage of different libraries loaded.

Graph in Fig.11, shows number of bytes transferred for inbound and outbound UDP connections made by the worm sample. Graph shows for first connection, 111 Outbound UDP bytes and 889 Inbound UDP bytes.

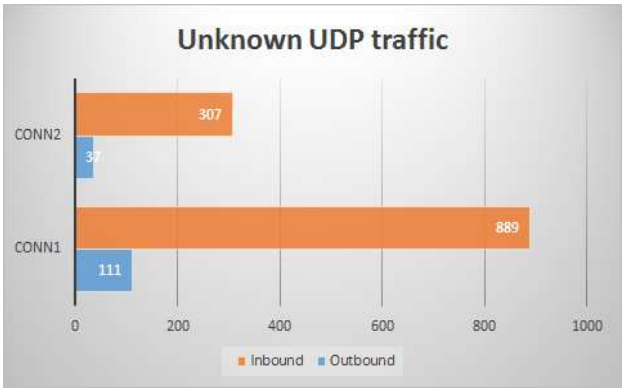

Fig.11. Inbound-Outbound Bytes Transfer

Fig.12, shows CPU and memory statistics recorded during the execution of worm sample.

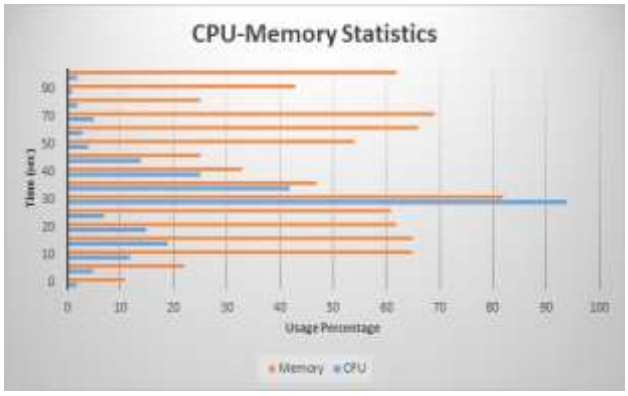

Fig.12. CPU and Memory Usage

V. Conclusions and Future Work

In this paper, a hybrid real-time zero-day malware analysis and reporting system is proposed. It aims to bridge the gap between zero-day malware detection and analysis by delivering the first inclusive behavioral report about a zero-day attack. It integrates various malware analysis tools and utilities in a component-based architecture where any of the function or utility can be replaced in the future. The SAE combines popular static tools and provides the basic information to profile the malicious binary. The DAE captures run-time behavior and has the capability to evade anti-analysis and anti-debugging checks of a malicious binary which may hinder the malware analysis process. Manual analysis is also intromitted to do step by step analysis of binary if needed. It also generates reports about zero-day malware behavior in HTML or PDF format.

In the future work it is planned to: (1) Achieve scalability and improve throughput of the system by analyzing multiple zero-day malwares at a time. (2) Automate full analysis process without any sort of human intervention. (3) To address multiple execution path problem in dynamic analysis by fuzzing different types of inputs.

References

[1] R. Kaur and M. Singh, “A Survey on Zero-Day Polymorphic Worm Detection Techniques”, in IEEE Communications Surveys & Tutorials, vol. 16, no. 3, pp. 1520-1549, March 2014.

[2] McAfee Labs, “McAfee threat report”. [Online] Available: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2014.pdf. [Accessed: May 2015].

[3] Panda Labs, “Panda Labs Threats Report”. [Online] Available: http://press.pandasecurity.com/wpcontent/uplo-ads/2014/05/Quaterly-PandaLabsreportQ1.pdf. [Accessed: May 2015].

[4] F. Y. Rashid, “How to detect zero-day malware and limit its impact”. [Online]. Available: http://www.darkreading.com- /attacksbreaches/how-to-detect-zero-day-malware-and-limit/240062798. [Accessed: May 2015]

[5] S. Kaur and M. Singh, “Automatic attack signature generation systems: A review”, vol. 11, no. 6, pp. 54-61, December 2013.

[6] J. Newsome and D. Song, “Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software”, 12th Annual Network and Distributed System Security Symposium (NDSS'05), February 2005.

[7] R. Perdiscia, W. Leea and N. Feamster, “Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces”, 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI'10), pp. 1-14, April 2010.

[8] M. Zubair Rafique and J. Caballero, "FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors", 16th International Symposium, RAID 2013, LNCS vol. 8145, pp. 144-163, October 2013.

[9] C. Azad, V.K. Jha, “Data Mining in Intrusion Detection: A Comparative Study of Methods, Types and Data Sets”, International Journal of Information Technology and Computer Science (IJITCS), vol.5, no.8, pp.75-90, 2013.

[10] M. Polychronakis, K. G. Anagnostakis and E. P. Markatos, “Network-level Polymorphic Shellcode Detection using Emulation”, in Journal in Computer Virology, vol. 2, no. 4, pp. 257-274, July 2006.

[11] M. Polychronakis, K. G. Anagnostakis and E. P. Markatos, “Emulation-based Detection of Non-self-contained Polymorphic Shellcode”, Proc. of the LNCS Springer 10th International Conference on Recent Advances in Intrusion Detection (RAID’07), Gold Goast, Australia, 2007, pp. 87-106.

[12] A. Abbasi, J. Wetzels, W. Bokslag, E. Zambon and S. Etalle, “On Emulation-Based Network Intrusion Detection Systems”, Proc. of the LNCS, Springer 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID’14), Gothenburg, Sweden, 2014, pp. 384-404.

[13] I. Santos, F. Brezo, X. Ugarte-Pedrero and P. G. Bringas, “Opcode sequences as representation of executables for data-mining-based unknown malware detection”, in Information Sciences, vol. 231, pp. 64–82, May 2013.

[14] H. Lu, X. Wang, B. Zhao, F. Wang and J. Su, “ENDMal: An anti-obfuscation and collaborative malware detection system using syscall sequences”, in Mathematical and Computer Modelling, vol. 58, no. 5, pp. 1140–1154, September 2013

[15] Y. Hou, J.W. Zhuge, D. Xin and W. Feng, “SBE - A Precise Shellcode Detection Engine Based on Emulation and Support Vector Machine”, Proc. of the LNCS, Springer 10th International Conference on Information Security Practice and Experience (ISPEC’14), Fuzhou, China, 2014, pp. 159-171.

[16] M. Zolotukhin and T. Hamalainen, “Detection of zero-day malware based on the analysis of opcode sequences”, Proc. of the IEEE 11th International Conference on Consumer Communications and Networking Conference (CCNC’14), Las Vegas, Nevada, USA, 2014, pp. 386-391.

[17] A. Lanzi and et. al., “AccessMiner: Using System-Centric Models for Malware Protection”, 17th ACM conference on Computer and communications security, pp. 399-412, October 2010.

[18] D. Mutz and et. al, “Anomalous System Call Detection”, ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 1, pp. 61-93, February 2006.

[19] A. Reina, A. Fattori and L. Cavallaro, “A System Call-Centric Analysis and Stimulation Technique Automatically Reconstruct Android Malware Behaviors”, 6th European Workshop on System Security (EUROSEC 2013), April 2013

[20] M. Sikorski and A. Honig, "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software", No Starch Press, February 2012.
[21] C. Eagle, "The IDA Pro Book, 2nd Edition- The Unofficial Guide to the World's Most Popular Disassembler", pp. 672, June 2011.
[22] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns". 12th USENIX Security Symposium, pp. 1–12, August 2003.
[23] M. Christodorescu, S. Jha, S. A. Seshia, D. X. Song, and R. E. Bryant. "Semantics aware malware detection". In IEEE Symposium on Security and Privacy, pp. 32–46, May 2005.
[24] H. Flake, "Structural comparison of executable objects". In Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'04), July 2004.
[25] A. Moser, C. Kruegel and E. Kirda, Limits of Static Analysis for Malware Detection, IEEE 23rd Annual Computer Security Applications Conference, Florida, pp. 421-430, December 2007.
[26] M. Sharif and et. al, Eureka: A Framework for Enabling Static Malware Analysis, 13th European Symposium on Research in Computer Security, Spain, pp. 481-500, October 2008.
[27] T. Dube and et. al, "Malware Target Recognition via Static Heuristics", Computers & Security, vol. 31, no. 1, pp. 137-147, Feburary 2012.
[28] F. Zhu and J. Wei, "Static Analysis based Invariant Detection for Commodity Operating Systems", Computers & Security, vol. 43, pp. 49-63, June 2014.
[29] M. Eskandari and S. Hashemi, "A Graph Mining Approach for Detecting Unknown Malware", Journal of Visual Languages and Computing, vol. 23, pp. 154-162, March 2012.
[30] U. Bayer and et al, "Dynamic Analysis of Malicious Code", Journal in Computer Virology, vol. 2, no. 1, pp. 66-77, May 2006.
[31] F. Bellard, "Qemu: A Fast and Portable Dynamic Translator", in USENIX Annual Technical Conference, pp. 1-41, April 2005.
[32] Cuckoo Sandbox, "Open Source Automated Malware Analysis", [Online] Available: https://media.blackhat.com/us-13/US-13-Bremer-Mo-Malware-Mo-Problems-Cuckoo-Sandbox-WP.pdf, August 2013. [Accessed, Jan 2015]
[33] C. Willems, T. Holz and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox", IEEE Security and Privacy, vol. 5, no. 2, pp. 32-39, April 2007.
[34] Norman Sandbox, [Online] http://download01.norman.no-/product_sheets/eng/SandBox_analyzer.pdf. [Accessed, Jan 2015]
[35] Joe Sandbox Technology, [Online] http://www.joe-security.org/joe-sandbox-technology. [Accessed, Jan 2015]
[36] VirusTotal, "Public API v2.0", [Online] https://www.vi-rustotal.com/en/documentation/public-api/. [Accessed, Jan 2015]
[37] M. Egele and et. al, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools", ACM Computing Surveys (CSUR), vol. 44, no. 2, pp. 1-49, February 2012.
[38] S. Sarkar, M. Brindha, "High Performance Network Security Using NIDS Approach", International Journal of Information Technology and Computer Science (IJITCS), vol.6, no.7, pp.47-55, July 2014.
[39] R. Wason, A.K. Soni, M. Qasim Rafiq, "Estimating Software Reliability by Monitoring Software Execution through OpCode", International Journal of Information Technology and Computer Science (IJITCS), vol.7, no.9, pp.23-30, April 2015.

Authors' Profiles

Ratinder Kaur is a PhD scholar at Thapar University carrying out her research in the field of Network Security. She holds strong academic record. She received her Bachelor's Degree from Punjab Technical University and holds a Master's Degree, with honors in Software Engineering from Thapar University. She showcases strong inclination towards Computer Security field which is evident from her master thesis on Operating System fingerprinting, for which she won TCS (Tata Consultancy Services) Best Student Project Award, and now exploring Zero-day attack frontiers.

Maninder Singh received his Bachelor's Degree from Pune University in 1994, and holds a Master's Degree, with honors in Software Engineering from Thapar Institute of Engineering & Technology, as well as a Doctoral Degree specialization in Network Security from Thapar University. He is currently working as Associate Professor in Computer Science and Engineering Department at Thapar University. Dr. Singh is on the Roll-of-honour at EC-Council USA, being certified as Ethical Hacker (C|EH), Security Analyst (ECSA) and Licensed Penetration Tester (LPT). Dr. Singh has successfully completed many consultancy projects for renowned national bank(s). His research interests include network security and grid computing, and he is a torchbearer for the open source community. He can be reached at, msingh@thapar.edu.

How to cite this paper: Ratinder Kaur, Maninder Singh,"Hybrid Real-time Zero-day Malware Analysis and Reporting System", International Journal of Information Technology and Computer Science(IJITCS), Vol.8, No.4, pp.63-73, 2016. DOI: 10.5815/ijitcs.2016.04.08