2016 IEEE 30th International Conference on Advanced Information Networking and Applications

A Cloud-Based Energy Efficient System for Enhancing the Detection and Prevention of Modern Malware

Qublai Khan Ali Mirza (1st),
Ghulam Mohi-ud-din (2nd)
School of Electrical Engineering and Computer Science
University of Bradford
Bradford, UK
q.k.alimirza@bradford.ac.uk (1st)

Irfan Awan (3rd)
School of Electrical Engineering and Computer Science
University of Bradford
Bradford, UK
i.u.awan@bradford.ac.uk

***Abstract*—in today's modern world, a simple malware attack can result catastrophically and can cause havoc. In spite of numerous types of antiviruses available in the market, there is a dearth in detection techniques of these antiviruses. This paper proposes a complete system, which is a combination of conventional and new techniques for detecting malware. We first evaluate the antiviruses against 10,000+ malware samples to highlight their weaknesses and then propose, implement, and benchmark the cloud-based system against some defined parameters. We have tested the effectiveness and efficiency of the proposed system by monitoring the detection rate and processing power it consumes in order to operate in a host machine.**

Keywords—Cloud-based; malware; energy efficient; malware detection; cyber-attacks; antiviruses; vulnerabilities; PEFrame; PE files

I. INTRODUCTION

Enterprise and home users heavily rely on different types of antiviruses for securing their personal files. However, the increasing number of malware attacks on all types of users put a huge question mark on the effectiveness of these antiviruses. Antiviruses struggle to detect and prevent modern and advanced level attacks and as the vendors of these antiviruses are adding more features in their newer versions, requiring enhanced privileges, which makes the whole system more vulnerable against different types of attacks [1]. Every attack on an enterprise or an individual user, doesn't matter what type of attack it is, proves that the attacker is at least one step ahead of the security companies and their complex security software [2].

Usually when a malware's code is released publicly, thousands of variants of that malware are born and every single variant has its own behaviour [3]. Therefore, it is practically impossible to analyse behaviours of each and every malware and their thousands of variants in order to avoid getting infected [4]. Enterprise and general antiviruses use traditional detection and prevention techniques for these modern day malicious software, which are not only ineffective but also exposes the whole infrastructure with many vulnerabilities [5]. According to Norton founding members, their antivirus can only detect and prevent 45% of the attacks on a system [6].

The security software of the modern era use signature, heuristics, and anomaly, etc. based detection techniques against malware, these malware are getting intelligent and complex drastically [1]. Such techniques are quite effective against the known malware that have been identified previously and their signature already exists in the antivirus database but it's very difficult for an antivirus to detect a malicious piece of code with no prior record available in any database [7].

Modern day malware are very complex and intelligent pieces of code, they have the ability to mutate themselves and completely change and obfuscate their appearance to confuse antiviruses and analysis tools [4] [8]. Modern day malware also have the ability to generate their mutated version with the help of mutation engine, which resides in the body of the malware and continuously generates different versions of the same malware [9].

A. Current Weaknesses & Limitations

Malware authors are using extremely efficient and innovative methodologies to create highly destructive malware. Conventional methodologies are not enough to identify and prevent these modern malware from infecting an enterprise or home user [10]. Security software, specifically modern antiviruses have increased complexity, require enhanced /administrative privileges, and consume a big chunk of computing resources. These complexities not only occupy more space on the host machine, they can also expose the host machine or the entire network to numerous vulnerabilities [11], which are quite commonly exploited by the malware authors. The dearth in detection techniques in antiviruses is inevitable, specially, when the losses in cyber-attacks are worth more than $600 billion dollars a year [12].

A new system or framework cannot be proposed until and unless the weaknesses and limitations of current systems are not discussed. Table 1 presents the evaluation results of 10 antiviruses against a small sample of 21 malware. Although, the malware set used in this initial analysis was really small and doesn't show a comprehensive evaluation but it does highlight that there is a problem, which should be solved in the most efficient way. Evaluation results highlight some serious

1550-445X/16 $31.00 © 2016 IEEE
DOI 10.1109/AINA.2016.133

concerns, even from this small sample, the top rated antiviruses such as Bitdefender, Kaspersky, McAffee weren't able to classify all of the malware as a threat to the system.

Table 1: Evaluation Result of 10 Antiviruses

No.	Antivirus Program	Detected Malware	Un-Detected Malware	CPU Usage	Total Samples
1	Bitdefender	17	4	30%	21
2	Kaspersky	15	6	47%	21
3	Webroot	20	1	18%	21
4	Emsisoft	17	4	39%	21
5	F-Secure	17	4	20%	21
6	MalwareBytes	12	9	21%	21
7	McAffee	14	7	34%	21
8	TrendMicro	16	5	37%	21
9	Panda	16	5	20%	21
10	VoodooSoft	18	3	17%	21

Additionally, if we look at the CPU usage of these antiviruses, it shows that they are consuming a really big chunk from the CPU resources while running. Usually for a home user, the schedule scan runs every day and the run time is directly proportional to the size of the file system, which means, for bigger file systems the antivirus take a longer time to finish its usual scanning and during that time it utilizes an average of 40% from the CPU resources [13]. To summarize the evaluation result, there is a requirement for an efficient system that doesn't utilizes the resources of the host machine and detects the malware efficiently and accurately.

Many modern antiviruses utilize the power of cloud computing for running their detection engines to avoid utilizing the CPU power of the host machine, which is a very good approach and can be enhanced for many proactive approaches in malware detection [11].

This paper proposes a new security system for malware detection, which is not only better in detection and prevention as compare to the leading antiviruses but also extremely efficient in consuming the host resources for doing the assigned tasks. Following are the main modules of the security system proposed in this paper:

- **Cloud Engine:** Cloud engine is comprised of a detection engine based on 15 antivirus engines, malware analysis module comprised of static analysis tool, and CTI (Cyber Threat Intelligence) data collection module.
- **Local Agent:** An efficient and lightweight agent for the host machine that detects malicious file and checks if the file is malicious or not by contacting the cloud engine. It also stores cache of files previously analyzed to straightaway detect similar threats.

II. Related Work

This section gives a synopsis of the related work, how it's relevant, weaknesses, and if it can be used in order to enhance our solution. One of the popular research projects from University of Michigan, titled: CloudAV, introduced the novel approach of having the N-version antivirus in the cloud with a light-weight network agent [2]. The edge this specific research had was the implementation on the real university network, which tested the whole system against the real-time data. Although, CloudAV was quite resilient against malicious attacks as compare to the traditional antiviruses it wasn't analyzing the suspicious file further if the antivirus engine was unable to classify it as malicious. This raises a serious question on the performance of the whole system because as discussed in the previous section antiviruses alone are unable to detect modern malicious attacks. Another feature CloudAV claimed to have was forensics information and retrospective detection, which could be extremely useful for the long run if they were storing the forensics information in a standardized way [2]. Their approach was quite efficient for the samples they have analyzed with the antivirus engine, which raises two serious issues; 1) how detailed was the forensic data, because it was only based on the feedback from their cloud antivirus engine. 2) As it was not stored in a standardized way, which means it cannot share that information with third parties neither can they get the forensic information from other security companies to enhance their dataset.

Antiviruses such as Avira and Panda are also utilizing the power of cloud by sending the processing jobs to the cloud-based detection engine [14]. As evaluation results presented in the later section of this paper illustrate that they are not very effective and do utilize the CPU resources and the detection rate is not as high as they claim.

III. Proposed Solution

In the previous section we discussed about different types of weaknesses and limitations present in antiviruses. Evaluation of 10 different antiviruses was presented, which was based on different parameters.

From all the problems identified above, this paper focuses on the efficiency of malware detection and CPU resource consumption. We then propose and implement a solution, which is tested on the same malware samples used to evaluate the antiviruses listed above.

1) Open Source Components

There are some external components, which we integrated with the system, mainly; publicly available APIs of 15 antiviruses, along with open source static analysis tool PEframe, and the OpenIOC standard for CTI (Cyber Threat Intelligence) data. The reason why only these tools and standards were used is explained below.

CTI (Cyber Threat Intelligence) data is quite effectively used by numerous cyber security companies these days. CTI data holds different types of information related to a breach in a system or in a network. There are multiple standards for storing such information such as; OpenIOC, CybOX, STIX, TAXII, TLP, OTX, etc. In this system, we are storing the information in the OpenIOC format [15].

OpenIOC is an XML based framework, which provides a flexible and well organized way of sharing the complex semantics of a malware [16]. Additionally, XML facilitates a well-recognized standard format of encoding information into a machine readable format this is why OpenIOC becomes widely acceptable as a Cyber Threat Intelligence (CTI) information sharing framework [15]. The main motivation

behind the creation of such framework is to find a common and widely understandable language to illustrate intrusion information and other cyber events. OpenIOC describes the low level attributes with the help of its comprehensive set of terminologies that can be easily interpreted into machine-understandable parameters [17]. These parameters and attributes can then be used as a configurational asset for numerous IT security and monitoring tools such as; IDS (Intrusion Detection System), anti-virus, IPS (Intrusion Prevention System), firewalls, etc [18]. The outputs and logs from these tools can be interpreted into OpenIOC documents that can be shared amongst other systems.

This project uses OpenIOC to store the threat intelligence information in a standardized manner, which can be later used by the system itself and the similar information available from third parties about numerous new malware can be integrated with the system's database to enhance the dataset

The proposed solution has a malware analysis module, which comprehensively analyzes the suspected file. This functionality

Table 2: Static Analysis Tools - Evaluation

Tool/Functions	Strings	PEiD	Exeinfo	Language 2000	Dependency Walker	PEView	Resrouce Hacker	PEBrowse	PE Explorer	PEStudio	PEFram
Command line	✓	•	•	•	•	•	•	•	•	•	✓
Open source	•	•	•	•	•	•	•	•	•	•	✓
File Metadata	•	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Resoruce info	•	•	•	•	•	•	✓	•	✓	•	•
Functions/DLLs	✓	•	•	•	✓	•	•	•	•	✓	✓
Indicators	•	•	•	•	•	•	•	•	•	✓	✓
VIRUSTOTAL	•	•	•	•	•	•	•	•	•	✓	•
Anti VM	•	•	•	•	•	•	•	•	•	•	✓
Anti Debug	•	•	•	•	•	•	•	•	•	•	✓
Packer Detection	•	•	✓	✓	✓	✓	•	•	✓	•	✓
Auto. Unpacking	•	•	•	•	•	•	•	•	✓	•	•
Auto Analysis	•	•	•	•	•	•	•	•	•	•	✓
Disassembly	•	•	•	•	•	•	•	✓	✓	•	•

is achieved by integrating an open source static analysis tool; PEframe. We evaluated numerous static analysis tools and based on the results and the parameters they were returning, we selected PEframe. Table 2 presents the evaluation of all the static analysis tools which were tested. It can be observed by going through the information presented in table 2, that PEframe gives the most relevant parameters in when an analysis is performed. The most important parameters, which is relevant for our system are Anti VM and Anti Debug. These two parameters identify, if the analyzed file has an anti-virtual machine and anti-debugging feature, which is quite common in modern day malware samples. This also allows to shrink down the number of false positives in the long run. Additionally, PEframe has two very important feature; a) it is command line based, b) it is open source, which makes it really convenient to integrate it with any system.

The proposed solution is divided into two main modules a) lightweight host agent & b) Cloud-based detection engine. These modules are connected to each other and work in a client-server architecture.

A. Lightweight Host Agent

As the name defines it, this host based module is designed to be extremely efficient in terms of consuming the CPU usage. This Host Agent runs as a background service on the host machine and it is comprised of the following modules; Detection module, Browser Extension, and the Cached Data module. All the submodules of this service are micro level and they have small but exclusive functionalities, which makes the whole system extremely efficient.

1) The Detection Module:

The detection module monitors the system and the files already stored and running on the machine. This module at the moment has a limited functionality and can only detect the .exe files executed by the host. To avoid repetition and usage of CPU resources, it only detects the unique files that are executed and ignore any repeated execution.

2) Browser Extension:

The browser extension is developed for Chrome and Firefox, which can be added to these browsers and then it checks every file downloaded on the machine before it becomes available to the user. The file is either checked locally or sent to the cloud-based detection engine for detection and analysis. This module is active only when the browser is running.

3) Cached Data Module:

The analysis report generated from the cloud-based detection engine is cached locally on the host machine. To detect and eliminate any threat independently and efficiently, the suspected file is matched with the local cached data and is eliminated immediately. If the file is not detected as a threat locally then it is sent to the cloud engine where it is further analyzed comprehensively and the threat data is updated on the server and on the host machine if the file is classified as malicious or even if it's not malicious.

This module doesn't rely on the processing power of the network or the host machine, as it is based on rich cloud-based server, which gives the whole system an edge over conventional antiviruses.

B. Cloud-Based Detection Engine

The cloud based detection engine is the backbone of the whole system, it is further divided into multiple modules working in an autonomous manner in a sequential model. Each subsequent module is triggered based on the output of the previous module, for e.g. if the output is 0 the next module will be triggered and if the output is 1 the operational module will return the result to the client and update the threat data on both client and server. This cloud-based engine is comprised of the following submodules: the detection engine (based on 15 antivirus engines), malware analysis module, and data collection module.

1) The Detection Engine

Although, antivirus, when working independently to protect a system is not enough to prevent it from getting infected but if multiple detection engines of powerful antiviruses are merged together to detect a malicious file then definitely the results will get better. This detection engine is comprised of 15 antivirus engines deployed on the cloud in a single module to detect the

suspected file sent by the client. When this module detects a malicious file it blocks the file immediately on the client and updates the threat data on the client and on the server. If it is unable to classify a file as a potential threat then it is transferred to the next module, which is the malware analysis module for further detailed analysis.

2) Malware Analysis Module

To further enhance the functionality of the system and to make it more resilient against advance level malicious attacks, this module works to support the system by further dissecting the potential threat and performing the static analysis of the suspected file. Control is transferred to this module by the detection engine when it is unable to detect the file as a threat. The result attained after the analysis of the suspected file classify the file as malicious or non-malicious and the analysis report is generated in a standardized OpenIOC format and stored in the data collection module.

3) Data Collection Module

The data collection module accumulates and stores the CTI (Cyber Threat Intelligence) data acquired from the detection engine and the malware analysis module in an OpenIOC standardized format. The data collection module plays vital role in the efficiency of the whole system as it can assist in detecting malicious files without any processing. Another advantage of using OpenIOC standards is that the dataset can be enhanced by accumulating the dataset of malicious files from third parties.

IV. System Architecture

This section gives an insight of the complete architecture of the system and how each module is placed to achieve the best possible result. As illustrated in figure 1, there are two main modules in the system; the first one is the lightweight host agent and the second one is the detection engine which resides in the cloud. Both these main modules are connected with each other with a communication module and different types of messages are transported through the communication module. Figure 1 is the high-level architecture of the system, which represents the complete functionality of the system. It can be observed that there is internal communication in both the modules, the host

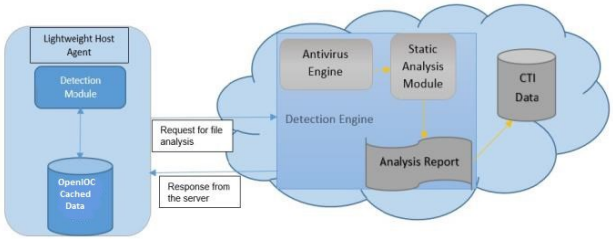

Figure 1: Complete System Architecture

agent attempts to detect the malicious file with the help of cached data stored on the host. If the detection module is unable to classify the file as malicious, it sends the file to the cloud-based detection engine for further investigation. The suspected file is received by the antivirus engine and straightaway it is analyzed by 15 antiviruses running concurrently on multiple cloud instances. If the antivirus engine is able to classify the file as malicious, it responds the host agent with a task to block the file along with an analysis report in the format of OpenIOC, which is stored on the CTI data collection module on the server and a copy is stored on the host machine for future reference. If the antivirus engine is unable to classify the file as malicious, it forwards the task to the static analysis module, which thoroughly analyzes the file and classify the file either as malicious or non-malicious. The report generated by the static analysis module is integrated with the report generated by the antivirus engine and stored in the CTI data module on the server and the verdict along with the CTI data is sent to the host agent. The whole process of detection is triggered by the host agent when it initiates the whole cycle by retrieving a unique executable or a recently downloaded file. As figure 2 elucidates, that process of detection can be triggered by either of the nodes; a) browser extension b) process log monitor. The

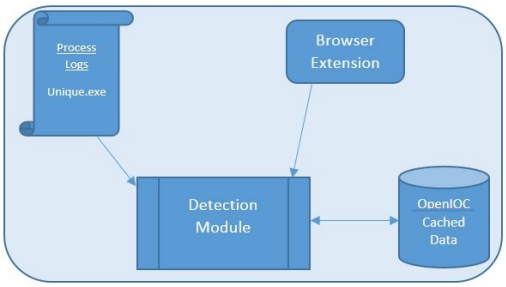

Figure 2: Lightweight Host Agent

browser extension sends the downloaded file to the detection module before it gets available to the user and the process log monitor, scans through all the processes and gets only the unique processes that are not scanned before. The detection module then checks the file against the OpenIOC-based CTI data cached locally. The retrieval of only unique executables from the process log makes the system really efficient by achieving the functionality of at least once and at most once at the same time.

V. Implementation

Previous section presented the understanding of the complete system architecture and how different modules are connected to each other. This section delivers a comprehensive and formal discussion about the implementation details of the complete system.

The implementation of any system is always based on some set of rules, techniques, or algorithms. Similarly, the implementation of the system proposed in this paper is also based on a set of algorithms, which are discussed in this section.

Since the architecture of the system is based on the following algorithm, therefore, it can easily be understood that the algorithm is divided into two parts. The first part of the algorithm defines the process initialization at the host machine, where lightweight host agent work and the second part shows the functionality of the cloud-based detection engine. Pseudocode 1 is the simplified version of algorithm 1, which

```
PROGRAM LightWeightAgent
F ⟵ Set of files downloaded
H ⟵ Local hash table
for every file f in F do
h ⟵ CalculateHash(f)
if h ∈ H
        if f ∈ Malicious set
                Mark file as unsafe
        else
                Mark file as safe
else
        Send file to cloud
        Save hash to Hash table
ENDFOR
WaitForResults();
END
```

Pseudocode 1: Lightweight Host Agent

defines the functionality of the lightweight agent. When a file is downloaded by the user, it is stored in F, which is a set of downloaded files and then the hash of that file is calculated and matched against H, which is local hash table. If hash of the downloaded file belongs to H then the file is classified as malicious, else, it is sent to the cloud engine for further analysis and the result is stored in the local cache.

Algorithm 1 defines the functionality of the host agent, where F represents the downloaded file on the system and $\mathcal{F}1$ represents the hash function, which extracts the hash value of the downloaded file represented as $\hbar$. If the extracted hash

$\hbar = \mathcal{F}_1$ (F)
$\hbar \in \mathbb{H} \rightarrow x = 1 \wedge \hbar \notin \mathbb{H} \rightarrow x = 0$
$x = 1$ & $x \notin \mathcal{M}$
$\rightarrow \boldsymbol{\mathcal{B}} \cup \{F\}$
$x = 0 \rightarrow \boldsymbol{\mathcal{CF}} \cup \{F\}$

Algorithm 1: Algorithm for lightweight Host Agent

value belongs to the hash value cache ($\mathbb{H}$) on the client $\hbar \in \mathbb{H}$, then the variable x is assigned the value 1. If $\hbar \notin \mathbb{H}$ then x is assigned 0. While x is 1, means that it belongs to the hash dataset but it's not from the hash dataset of $\mathcal{M}$, which represents the malware cache stored on the host, then the file is from trusted programs $\boldsymbol{\mathcal{B}}$ and host machine can execute it. If x is 0, which means the file cannot be checked locally, then $x = 0 \rightarrow \boldsymbol{\mathcal{CF}} \cup \{F\}$, file is sent to the cloud engine, $\boldsymbol{\mathcal{CF}}$ represent the file in cloud engine.

Pseudocode 2 presents the simplified version of Algorithm 2, which defines the functionality of the cloud-based engine.

```
PROGRAM CloudModule
C ⟵ Set of files in Cloud
for every f in C do
    while i approaches to n
        Ri = Ai(f)
    endwhile
    for every R in Ri do
      AnalysisResult1 = ApplyMaliciousProperties(R)
    endfor
    If AnalysisResult1 is positive
      while i approaches to n
        Ri = Si(f)
      endwhile
      for every R in Ri do
        AnalysisResult2 = ApplyMaliciousProperties(R)
      endfor
      If AnalysisResult1 is positive
        Mark file as safe
      else
        Mark file as unsafe
   else
     Mark file as unsafe
endfor
CTI = ProduceCTI(AnalysisResult1, AnalysisResult2,
IndicatorTerms)
SaveAndShareCTI()
END
```

Pseudocode 2: Cloud-Based Detection Engine

This pseudocode shows the initiation of the cloud engine when the file is received from the local agent for further analysis. The antivirus engine runs the file fifteen times as a loop to get analysis report from all the fifteen antiviruses embedded in the cloud engine. If the antivirus engine is unable to classify the file as malicious then it transfer the

$\sum^{n}_{i=1} \boldsymbol{\mathcal{A}} \text{Ŕ}_i = \sum^{n}_{i=1} \boldsymbol{\mathcal{A}} \text{E}_{\text{ŕ}i}\,(\boldsymbol{\mathcal{CF}})$
$\acute{O}_1 = \sum^{n}_{i=1} \mathcal{F}\,(\boldsymbol{\mathcal{A}} \text{Ŕ}_1\,, \mathbb{C})$
$\acute{O}_1 \rightarrow \sum^{n}_{i=1} \acute{S}\text{Ŕ}_i = \sum^{n}_{i=1} \acute{S}\,\check{T}_{i\,(}\boldsymbol{\mathcal{CF}})$
$\acute{O}_2 = \sum^{n}_{i=1} \mathcal{F}\,(\,\acute{S}\text{Ŕ}_i\,, \mathbb{C})$
$\acute{O}_2 \rightarrow \boldsymbol{\mathcal{A}} \cup \{\boldsymbol{\mathcal{CF}}\}$
$\neg\, \acute{O}_2 \rightarrow \boldsymbol{\mathcal{B}} \cup \{\boldsymbol{\mathcal{CF}}\}$
$\neg\, \acute{O}_1 \rightarrow \boldsymbol{\mathcal{B}} \cup \{\boldsymbol{\mathcal{CF}}\}$
$\{t: t \in \boldsymbol{\mathcal{T}} \rightarrow (t \cap \acute{O}_1) \cup (t \cap \acute{O}_2)\}$
$\text{Ŕ}_1 = \{(t \cap \acute{O}_1) \cup (t \cap \acute{O}_2)\}$

Algorithm 2: Algorithm for Cloud-Based Detection Engine

control of the program to the static analysis module for an in-depth analysis of the suspected file.

Algorithm 2 defines the functionality of the cloud-based detection engine. $\sum^{n}_{i=1} \boldsymbol{\mathcal{A}} \text{Ŕ}_{i,}$ represents the loop in which, the

suspected file is analyzed. Where n is the number of antivirus engines, which is fifteen and $\mathcal{AR}$ is the result of antivirus analysis, which is generated by each engine separately. This means that $\sum^{n}_{i=1} \mathcal{AR}_i$ gives fifteen different analysis reports generated by each engine. Therefore, $\sum^{n}_{i=1} \mathcal{AE}_i(\mathcal{CF})$ means that function $\mathcal{AE}_i$, which represents antivirus analysis function is performed on $\mathcal{CF}$ fifteen times. $\sum^{n}_{i=1} \mathcal{AR}_i = \sum^{n}_{i=1} \mathcal{AE}_i(\mathcal{CF})$ this complete equation defines that file is analyzed fifteen times and the analysis report of fifteen antivirus engines is stored each time the analysis is performed. Once the analysis of antivirus engine is completed then the reports are saved into output 1 represented as $Ó_1$ and function $\mathcal{F}(\mathcal{AR}_1, \mathbb{C})$ is applied on the report, which means, constraints $\mathbb{C}$ are checked in the antivirus analysis result $\mathcal{AR}_1$, checking whether the file is packed or unpacked, malicious or non-malicious, etc. if $Ó_1$ is classified as non-malicious then the static analysis module is triggered. $Ó_1 \rightarrow \sum^{n}_{i=1} ŚR_i = \sum^{n}_{i=1} ŚŤ_i(\mathcal{CF})$, represent that static analysis tool function $ŚŤ_i$ is applied on the suspected file and the result of the analysis is again stored in $Ó_2$. The algorithm is quite flexible and $\sum^{n}_{i=1}$ in the static analysis function represents that more static analysis tools can be added in the module and the system will still work smoothly. After applying the function of constraints on the static analysis results $\mathcal{F}(ŚR_i, \mathbb{C})$ if the file matches the constraints of malicious program it is then classified as malicious $Ó_2 \rightarrow \mathcal{A} \cup \{\mathcal{CF}\}$. If the file doesn't satisfy the constraints for malicious program then is classified as safe $\neg Ó_1 \rightarrow \mathcal{B} \cup \{\mathcal{CF}\}$ and whatever the output of the analysis is the data collection module, which holds the CTI data in OpenIOC format is updated $\{t: t \in \mathcal{T} \rightarrow (t \cap Ó_1) \cup (t \cap Ó_2)\}$. The above discussion presents the complete mechanisms of the whole system and how the module communicate at the lower level and what rules and constraints are considered.

VI. Results

This section provides the results of evaluation of different

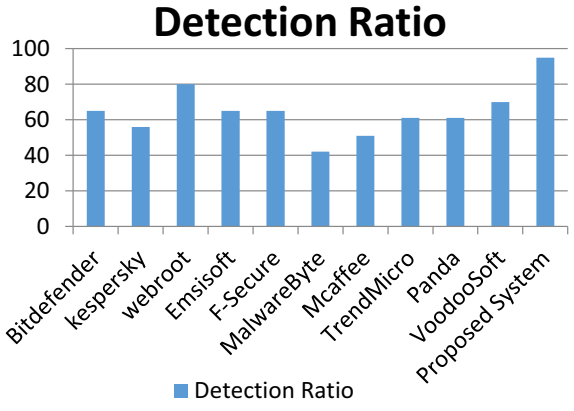

Figure 3: Evaluation of Antiviruses

antiviruses against a sample of ten thousands malware and then it discusses the results of the proposed system after matching them with the top ten antiviruses.

For the evaluation of different antiviruses, we chose ten antiviruses and tested them against a set of ten thousand malware samples. Figure 3 presents a bar graph of detection ratio of malware by antivirus programs and the proposed system. There is a huge difference between table 1, which

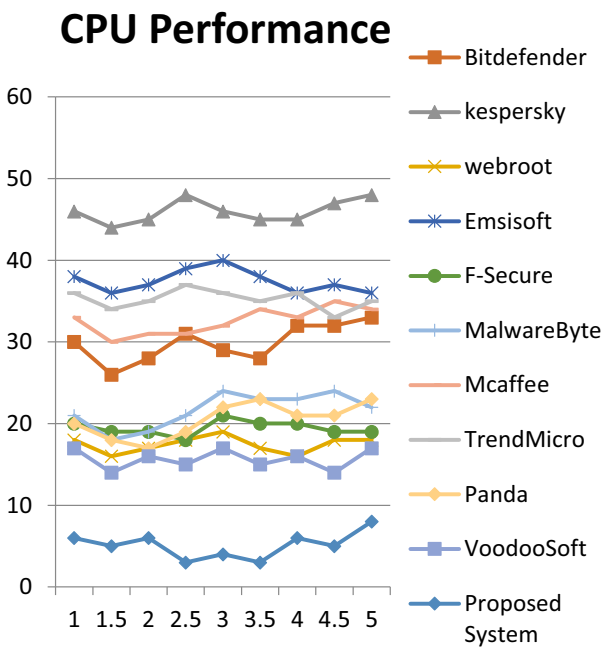

Figure 4: CPU Usage Graph of Antiviruses

presents the initial evaluation details of the same antiviruses against twenty one malware samples and figure 3, which presents the evaluation result of the same antiviruses against a set of ten thousand malware samples. The difference between

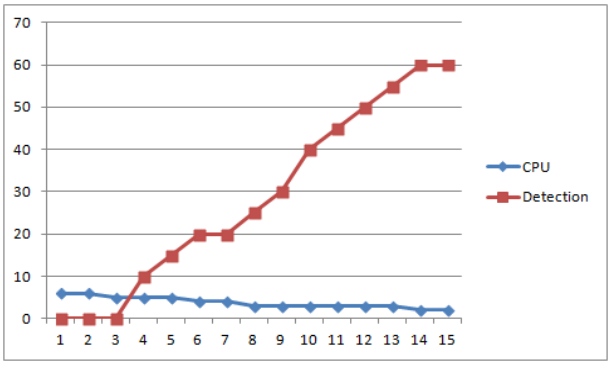

Figure 5: Evaluation of Lightweight Host Agent

these two evaluations is quite clear, as it can be observed that the detection ratio of these antiviruses decreases when the number of samples is increased. Additionally, table 1 presents the CPU consumption of these antiviruses while they are running, which is fairly high. Figure 3 and figure 4 also illustrate the efficiency and accuracy of the proposed system. The proposed system detects 98% of the malicious files and consumes only a maximum 6% CPU. The closest competitor of

the proposed system if Webroot, which detected 80% of the malicious file and consumed around 19% of the CPU. From the remaining antiviruses, Voodoosoft detected 70% of the malicious files and consumed 20% of the CPU power. The astonishing result came from Bitdefender, Mcafee, they detected 65% and 50% of the malicious files and consumed a big chunk of the CPU, around 35% each. As discussed in the earlier section about the claims made by antiviruses such as Panda and Avira that they detect 99.9% of the malware and utilize the power of the cloud to run the intensive tasks. It can also be observed from figure 3 & 4 that Panda detects 78% of the malware and consumes 22% of the CPU while running.

Figure 5 presents the evaluation result of the lightweight host agent from the deployment till the complete analysis of ten thousand samples. The x-axis of the graph is the time in hours and y-axis represent the detection percentage. It can be observed from figure 5 that local detection was 0% for the first 3 hours from deployment, lightweight agent didn't detect anything itself and sent every suspected file to the cloud engine and the CPU consumption was 6%. After 3 hours the local detection ratio went up drastically, which is because for the first 3 hours the local cache was populating by retrieving the analysis reports from the cloud engine. Once the local cache got rich enough to detect locally it started to match and detect the files accurately. As the time passed the detection ratio was further enhanced and additionally, the CPU consumption dropped as well to just 2%. The evaluation of lightweight agent is very useful to identify the performance of the complete system, because the local cache is based on the rich analysis reports retrieved from the cloud engine, which will enhance significantly as more and more files are analyzed.

This section discussed the results acquired by various experiments performed on the proposed system and ten different antiviruses. The results prove that the proposed system has a better detection rate around 99.9% and consumes maximum 6% of the CPU while running to its full potential. This not only proves the effectiveness of the proposed system but it also satisfy the claim of energy efficiency made in the start of this paper.

VII. Limitations

Although, the claims made and the results presented in the previous sections, determine that the discussed system is highly robust and efficient but there are some limitations in the system, which doesn't allow it to perform to its full potential.

The open source static analysis tool integrated in the cloud engine for further investigation of the file only analyzes the PE files, which is adequate for windows based systems but this tool is not able to cater the requests from Mac or Linux based machines. This restricts the system to work with its full potential for only windows-based machines.

Another major limitation currently faced by the system is the limited functionality of the host agent, which cannot detect any malicious file stored on the system until it executes and appears in the process log, which leaves the system vulnerable because it could be used to transport malicious files rather than executing them.

The proposed system and specifically the host agent is currently designed for a single windows-based machine and cannot be deployed in a network environment, which limits the potential of the system because the current server implementation is powerful enough to cater a windows-based enterprise network. The current dataset is based on 10,000 malware samples, which is not very comprehensive but the system is deployed on multiple machines and it is continuously populating its dataset. This will make the system stronger and resilient after each day. The further discussion on how these above mentioned limitations can be diminished and how the system can be further enhanced is presented in the next section.

VIII. Future Work

The discussion on results and evaluations prove the efficiency of the system in all the domains targeted in the beginning. The system performs as claimed earlier but to enhance the performance of the system and to make it scalable for enterprise networks and non-windows-based environments, it requires some extensions and modifications.

As discussed earlier, the static analysis tool integrated with the cloud engine only analyzes portable executable files, which makes it exclusive for only windows based files. This can be enhanced by adding another static analysis tool in conjunction with PEframe, which will make the whole cloud-based engine powerful enough to analyze and detect malicious files from any type of host machine or network and it will enhance the heterogeneity of the whole system.

Another enhancement, which will categorically make the system resilient and efficient is the addition of a module that could convert CTI information into OpenIOC format. The addition of such module will not only add value to the system but it will allow information to be shared with third parties that using different formats to store CTI data and CTI data from third parties can easily be integrated with our dataset to enhance the system performance.

One of the main enhancement, which could make the system a proper cyber-defense solution is the appropriate enhancement in the lightweight host agent. Currently, the functionality of this module is limited to single machine and even in the single machine not all the areas related to detection are covered. Enhancement such as adding a network crawler module, which learns the network and monitors any changes in the structure of the network will allow the system to monitor the network at uninterruptedly. Additionally, features of IDS (Intrusion Detection System) will complement the resilience of the system against numerous types of malicious attacks and it will protect the host machine or network more efficiently.

The complete system proposed in this paper is designed with an anticipation that the further enhancements discussed in this section will transform it into a complete solution for the enterprise network.

IX. Conclusion

To cater the problem of ever-enhancing ecosystem of malware and cyberattacks, we have proposed a new and efficient defense system, which uses the power of cloud computing to migrate

the process intensive tasks of malware detection to the cloud along with the addition of malware analysis module, which further enhances the detection by performing the static analysis of the suspected file. The detection engine in the cloud runs 15 antivirus engines in concurrent instances and if this engine is unable to classify the file as malicious, it triggers the malware analysis module for further investigation. This novel system delivers substantial amount of advantages over conventional antiviruses in terms of malware detection and energy efficiency. It uses maximum 6% of CPU power of the host machine while performing any detection task and storage of cached data on the host machine enhance the level of efficiency by detecting previously identified malware locally. Data collection module of the system, which stores the threat intelligence data in a standardized format of OpenIOC allows the integration of threat intelligence data from third parties, making the system more robust and reliable to detect any type of modern malware. We evaluated the efficiency of the proposed system and validated how it delivers considerably superior level fortification of end hosts against modern malware attacks.

X. REFERENCES

[1] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, "SplitScreen: Enabling efficient, distributed malware detection," *Commun. Netw. J. Of*, vol. 13, no. 2, pp. 187–200, 2011.

[2] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-Version Antivirus in the Network Cloud," University of Michigan, 2008.

[3] "A Brief History of Malware Obfuscation: Part 1 of 2," *blogs@Cisco - Cisco Blogs*. [Online]. Available: http://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2/. [Accessed: 05-Aug-2014].

[4] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," 2010, pp. 297–300.

[5] J. Bergeron and M. Debbabi, "Static Detection of Malicious Code in Executable Programs." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.6845&rep=rep1&type=pdf. [Accessed: 12-Aug-2015].

[6] Sky News, "Antivirus Is 'Dead' Says Norton Software Boss," *Sky News*, 06-May-2014. [Online]. Available: http://news.sky.com/story/1256251/antivirus-is-dead-says-norton-software-boss. [Accessed: 07-Jul-2015].

[7] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, 2008. DSN 2008*, 2008, pp. 177–186.

[8] M. Sikorski and A. Honig, *Practical malware analysis the hands-on guide to dissecting malicious software*. San Francisco: No Starch Press, 2012.

[9] D. Bruschi, M. Lorenzo, and M. Monga, "Detecting self-mutating malware using control-flow graph matching," in *Detection of Intrusions and Malware & Vulnerability Assessment*, vol. 4064, 2006, pp. 129–143.

[10] E. Chien, "Techniques of adware and spyware," in *the Proceedings of the Fifteenth Virus Bulletin Conference, Dublin Ireland*, 2005, vol. 47.

[11] M. Christodorescu and S. Jha, "Testing Malware Detectors," in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA, 2004, pp. 34–44.

[12] McAfee, "Net Losses: Estimating the Global Cost of Cybercrime," Center for Strategic and International Studies, Jun. 2014.

[13] M. Glover, "Fast virus scanning," US6763466 B1, 13-Jul-2004.

[14] "Avira Free Antivirus | Download the Best Free Antivirus Software," *Avira*. [Online]. Available: https://www.avira.com/en/avira-free-antivirus. [Accessed: 30-Oct-2015].

[15] OpenIOC, "Sophisticated Indicators for the Modern Threat Landscape: An Introduction to OpenIOC," OpenIOC, White Paper.

[16] D. Wilson, "The History of OpenIOC « Threat Research," *FireEye*, 17-Sep-2013. [Online]. Available: https://www.fireeye.com/blog/threat-research/2013/09/history-openioc.html. [Accessed: 31-Oct-2015].

[17] "Understanding Indicators of Compromise (IOC) Part I," *Speaking of Security - The RSA Blog and Podcast*. [Online]. Available: https://blogs.rsa.com/understanding-indicators-of-compromise-ioc-part-i/. [Accessed: 16-Nov-2014].

[18] Hun-Ya Lock and Adam Kliarsky, "Using IOC (Indicators of Compromise) in Malware Forensics," SANS Institute InfoSec Reading Room, Feb. 2013.