CloudIntell: An Intelligent Malware Detection System

Qublai K. Ali Mirza[1], Irfan Awan

School of Electrical Engineering and
Computer Science
University of Bradford, UK

Muhammad Younas

Computing and Communication Technologies
Oxford Brookes University, UK

Abstract

Enterprises and individual users heavily rely on the abilities of antiviruses and other security mechanisms. However, the methodologies used by such software are not enough to detect and prevent most of the malicious activities and also consume a huge amount of resources of the host machine for their regular operations. In this paper, we propose a combination of machine learning techniques applied on a rich set of features extracted from a large dataset of benign and malicious files through a bespoke feature extraction tool. We extracted a rich set of features from each file and applied support vector machine, decision tree, and boosting on decision tree to get the highest possible detection rate. We also introduce a cloud-based scalable architecture hosted on Amazon web services to cater the needs of detection methodology. We tested our methodology against different scenarios and generated high achieving results with lowest energy consumption of the host machine.

Keywords: Malware Analysis, Machine Learning, Cloud, Decision Tree, Boosting, SVM, Security

[1] q.k.alimirza@bradford.ac.uk(Corresponding Author)

1. Introduction

A recent report from Intel Security claims that they detect more than 7000 malware attacks every hour and this is only for mobile devices [1]. The number is significantly bigger if the domain is broader, such as; individual computers, enterprise networks, websites, and other web enabled devices and infrastructures. Figure 1 presents the statistics, published by AV-TEST, of malware released every year, AV-TEST institute registers near to 400,000 new malware every day [2]. Out of millions of malwares collected each year, majority of the samples are the evolved version of their predecessor, which happens because of an extremely intelligent yet malicious approach taken by the malware authors. When a malware author decides to release the code in public, many of them are released with a mutation engine for their [3]. This not only allows other people with malicious intent to generate their version of that malware, it also doesnt require a lot of programming knowledge for doing so. Even if only the original code is released by the malware author, it becomes easy for anyone to manipulate the code or add some more malicious features to use the malware for their own gain. This gives an enormous edge to the black-hat community over the white-hat community by saving time and recreating a sophisticated and more lethal version of an old malware. Malware authors use obfuscation and replication techniques to change the apparent features of malware dynamically to avoid getting detected by antiviruses and even if a single instance of a malware is detected, multiple, yet completely different, instances of the same malware are generated making it nearly impossible for the security software to detect it [4, 5].

The number of malware released every year is enormously more than previous years [6] and it is practically impossible to separately analyze the features and behavior of every malware released in the wild. Although, antiviruses claim to protect the individual systems, enterprise networks, emails, and other types of web-forgery, their progress is not as impressive as claimed by their vendors. Antiviruses not only fail to protect against modern sophisticated attacks, as

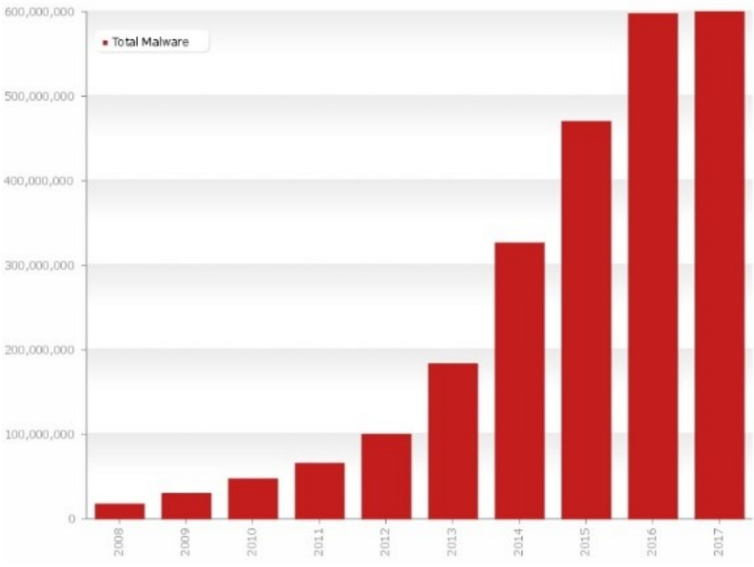


Figure 1: Malware Release Stats per Year by AVTEST [2]

proved by many researchers, they also consume a lot of resources for trying to do so. However, antiviruses and their detection strategy cannot be classified as useless, the techniques and methodologies used by antiviruses to detect and prevent malware from attacking and propagating are quite useful for malware that are known and whose signature and heuristics are present in antivirus's database [7]. Although, only relying on antiviruses is a huge mistake, as they consume a lot of resources and cannot completely secure the targeted machine or network. Therefore, many security companies and researchers have proposed and implemented alternatives for malware detection that deliver promising results.

Detecting a malware and preventing its infection or further propagation in a local network or in the wild requires an understanding of the techniques that are used by its authors, which includes a comprehensive understanding of all the apparent features of malicious files along with how they behave in a system or in a networked environment. The static and dynamic analysis of malware produce apparent and behavioral features of malicious files respectively, which allows the malware analysts and security experts to understand the dynamics of different types of vulnerabilities and the malware which exploit those vulnerabilities. Both these analysis techniques are useful in different scenarios but if the sole purpose is to reduce the time consumed during detection along with the resources consumption for doing so, then static analysis is a better choice. Analyzing a file statically doesnt guarantee that it is going to be perfectly identified as malicious or safe, it depends on the comprehensiveness of the analysis along with the relevance of the parameters that are considered to classify the file as safe or malicious. Moreover, for the system to understand whether the file is safe or malicious, it has to first learn about the apparent features of both the types. Therefore, it is important to apply existing and relevant machine learning algorithms to train the system. It will not only enhance systems awareness about malicious and legitimate files, it will reduce the time and resources consumed while detecting the malware.

This paper is an extension of an existing work published in [10], which mainly focused on two very important objectives; a) higher detection rate, b) low resource consumption. Higher detection rate [10] was achieved by combining the detection mechanism of ten antiviruses with a comprehensive open source static analysis tool, which gave a thorough output quite rapidly. Resource consumption was reduced by setting up the entire detection engine on a cloud-based server. The results from the above-mentioned approach were promising and achieved what was proposed initially.

The system developed and implemented in [10] was part of a pilot study, which had a primary objective of proving the hypothesis that a system can be developed that can consume very limited resources of the host machine to detect around 98% of the malicious files. Figure 2 presents the comparison of detection ratio between initial study [10] and major antiviruses. Although, the result does support the initial hypothesis but system does need a more resilient back end architecture supported and trained by some verified machine learning algorithms.

Understanding malware; their behavior, motives, and origins, is an infinite

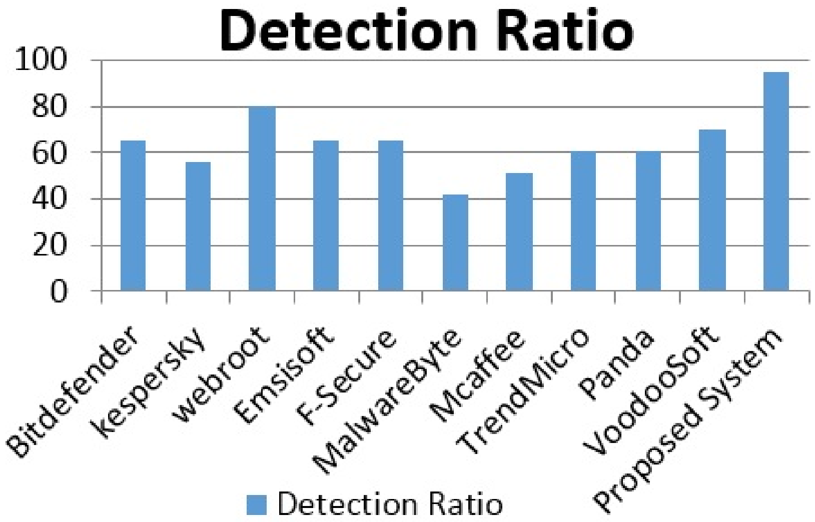


Figure 2: Detection Ratio of Initial Study [10]

paradigm and it is practically impossible to do that even by investing a massive amount of resources in the field. The recent news about an old attack [11] is a proof that the size of an organization is directly proportional to the level of destruction a cyberattack can cause. The attack on Yahoo, which they claim was initiated in 2013 and still not completely rectified [11], shows the intensity of a sophisticated attack on a large organization. Several security experts, using different manual detection techniques, to protect an organization, can be bypassed by a sophisticated attack. Whereas, an automated system simultaneously using multiple techniques, which includes; conventional detection, thorough analysis, and machine learning, can surely enhance the level of security for that organization and at the same time enhance its own performance with every iteration. However, such approach can only be successful if it is not burdening the enterprise infrastructure by consuming a lot of resources to make the system secure and resilient against malware attacks [12].

Large enterprise networks and even individual computers generate a large amount of network and process logs, which are analyzed by security analysts and administrators to detect any malicious behavior. These logs and similar data if analyzed properly can protect the system against many, if not every, type of attacks. Many researchers and corporate sector entities are incorporating machine learning for malware detection and to predict any future attacks. Machine learning can significantly enhance the detection mechanism by accurately detecting known attacks along with predicting any previously unknown malicious activity with very high true positive rate. CloudIntell, incorporates an optimum combination of machine learning algorithms that can efficiently detect a malicious activity without consuming a lot of resources of the client machine or network it is suppose to monitor.

Several different domains are benefiting from the scalable features of cloud platforms by utilizing services, infrastructures, platform applications, etc. offered by the cloud service providers. Migrating to the cloud or using cloud services allow businesses or applications to scale their scope or simply reduce their overall costs. CloudIntell uses several infrastructural services offered by AWS (Amazon Web Services) to enhance the energy efficiency initially proposed in the pilot study [10]. The pilot study [10] did utilize the cloud platform for energy efficiency but this extended version is designed to thoroughly utilize the cloud infrastructure. This will not only enhance the persistence in energy efficiency of the entire system, it will also help the analysis and detection mechanisms by scaling the system to take many requests simultaneously.

CloudIntell is more appealing and close to industrial implementation as compared to many similar approaches for the following reasons:

- Although, dynamic analysis has the ability to retrieve a huge amount of behavioral characteristics from a malicious file but the implications of this type of analysis include higher resource consumption along with analysts involvement in the process. The comprehensiveness and automation in static analysis techniques can generate a set of features that can be used to identify a malware with much lesser resources.

- Enterprise environment requires a system that is proactive and can detect any sophisticated attack before it can penetrate the network. Having machine learning techniques that could learn from the analysis of a huge number of malware can predict the behavior of a file without an in-depth analysis, consuming time and other resources.

- The entire architecture of this system is based on the idea of scalability. The utilization of the cloud platform not only serves as an idea of consuming less energy of the client machine it is monitoring, it also allows the system to be extended and used by different enterprises. As the system is based on a client server architecture, The client module works as a lightweight agent on the host machine and it is powered by the server module. This allows the client module to benefit from the rich server without consuming host machines CPU or networks resources.
- The comprehensive mechanism of classifying a file as malicious or safe, verifies the authenticity of the system along with the generation of detailed analysis reports, which can be used for real-time detection and prevention of known and unknown threats.
- Not many systems with such features generate output which can further be used for the enhancement of other systems. The analysis report generation in an appropriate and easy to understand manner could facilitate the sharing of threat intelligence data on a larger scale.

The rest of the paper is structured as follows; section 2 presents the related work done in the same area, section 3 presents the in-depth discussion of the architecture of the entire system and how it is implemented, section 4 presents the details of data collection for the experiments, section 5 presents the methodology used to enhance the detection of malware and reduce the resource consumption, along with the machine learning algorithms used, section 6 presents the malware detection experiments and their results, section 7 presents the discussion on the results achieved through experiments, and section 8 concludes the work discusses the possible future directions.

2. Related Work

Using machine learning for the identification of malware has been proposed using several different techniques by many researchers [13, 14, 15, 16, 17]. Each of these studies have their own methodologies to approach the problem of malware identification, by increasing the true positive, and reducing the false positive rate. Majority of the research in malware detection is based on windows-based malware and only focus on the detection of one type of malicious code. However, more than 90% of the industrial environment is based on windows, therefore, the threat of windows-based malware is significantly higher.

The conventional techniques of malware detection, also known as signature-based detection used by antiviruses are still quite useful and it can flawlessly detect a known malware. These techniques are not very helpful when there is an attack from a new or unknown malware, which is why there is a huge gap in the industry, despite several studies in this area.

One of the most relevant studies in this area were conducted by Kolter and Maloof (hereon KM) [13]. They drew techniques from machine learning and data mining and applied them on their collection. In their study [18], they used a common text classification practice, n-grams, which tested the results of various classifiers on malware detection. The techniques included in their research were; SVM, decision trees, Nave Bayes, and then applying boosting on each of the techniques. The KM approach used the AUC (Area under Curve) of an ROC (Receiver Operating Characteristic) to evaluate the performance of their classifier. They tested based on the highest information gains n-grams, computed with the help of the following equation:

$$IG(j) = \sum_{v_j \in \{0,1\}} \sum_{C_i} P(v_j, C_i) log \frac{P(v_j, C_i)}{P(v_j)P(C_i)} \tag{1}$$

They treated the presence or absence of the specified n-gram as Boolean on their classifier for boosted decision tree. As per their results, their model of boosted decision tree was able accomplish the finest accuracy rate out of all, achieving a 95% confidence interval AUC i.e. 0.9958 ± 0.0024. Boosting significantly enhances the performance of weak or unstable classifiers by decreasing their variance and bias but it can affect inversely on the stable classifiers [24], KM approach claims to improve the stable classifiers through boosting as well. The samples both benign and malicious used by them comprised of 1971 benign files and 1651 malicious files. The benign executables were retrieved from Windows OS (XP, 2000), and other online resources. Whereas, the malicious collection was obtained from MITRE Corporation and VX Heavens online repository. The KM research also used their approach of static heuristics technique for identifying payload functionality of malware. It identifies the functionality without dynamically analyzing malware, which is an efficient way because it doesnt utilize resources required for sandboxing and eliminates the threats involved in dynamic analysis. They were able to identify payload functionality with the help of reverse engineering analysis reports of a subset of their complete collection. The KM approach showed promising results in two different directions; malware detection and payload identification. However, there are some weaknesses in this approach, considering the small sample size, missing 6 out of 291 malicious files is a real game changer in real life detection. This means if the dataset is bigger, it can significantly increase the number of malicious files missed in a scan, which should be the main concern while detecting malware. Moreover, KM approach is not very effective for obfuscated malware and can easily omit such malicious files during the detection process.

MaTR [14] approach is another noteworthy contribution in this domain in which they recreated the experimental environment of KM using same dataset and the same formula presented in equation 1 to highlight their weaknesses. MaTR approach used 31193 malicious and 25195 clean files in the initial experiment and compared their results with KM approach, which showed improvements over KM with the following mean and confidence intervals.

The MaTR approach outperforms the KM approach and prove it by recreating

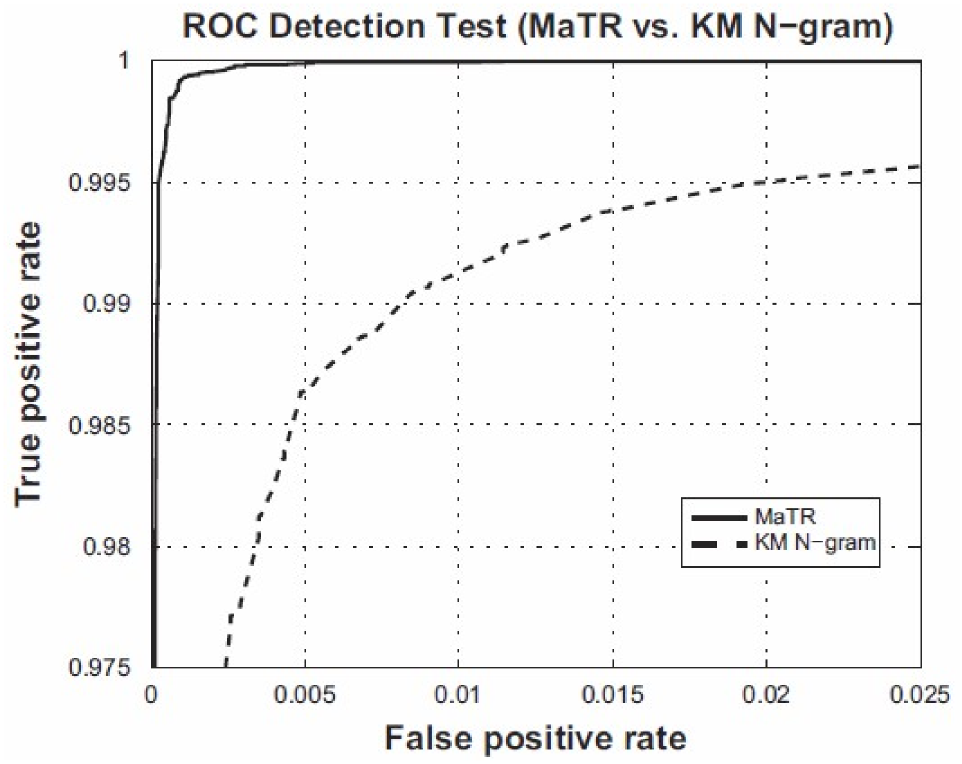


Figure 3: ROC Curves for KM n-gram Retest and MaTR [14]

the KM experiments, as presented in figure 3. MaTR system design introduces an interesting approach by adding a human component in its system as illustrated in figure 4. The reason behind introducing a human component is to give the system a capability of real-time detection. This means that with the help of a human operator continuously monitoring the system logs, decisions on legitimacy of the files can be made by looking at live anomalies occurring in the network. The human operator in this case provides appropriate responses for the type of malware rather than an automated and fixed response for all type of malware.

For the classification of malware, MaTR approach uses bagged decision tree classifiers, which can enhance the performance of simple decision tree and make

Method	Mean	95%CI
MaTR	0.999914	0.999840-0.999987
KM Retest	0.999173	0.998926-0.999421
KM Original	0.9958	0.9934-0.9982

Table 1: Mean AUC and Confidence Interval [14]

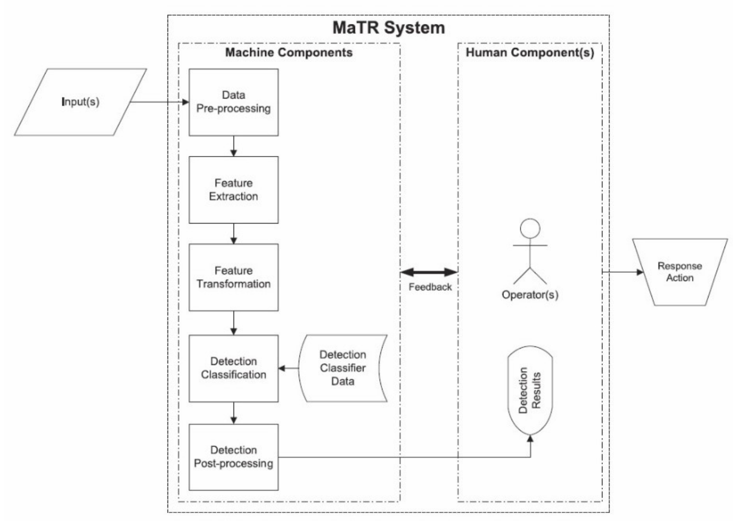


Figure 4: MaTR System Flow Diagram [14]

the results more accurate. The MaTR approach heavily relies on the human operator to take decision based on the detection results, which can be its main weakness. In the live environment, when a malware attacks a system or an enterprise network an automated response is necessary because when it comes to malware detection time is a key component. A malware can propagate and replicate itself within minutes inside a network or in a single machine, which means that an automated response is necessary. In MaTR approach, the parameter of response time and its affects are not mentioned. Additionally, the classification methodology of MaTR claims a very high detection rate, however, the performance on obfuscated malware is not present and it lacks the ability to do so.

3. System Architecture

One of the contribution of this paper is the energy efficiency, which is one of the weakest areas of many antiviruses. When it comes to higher performance versus energy efficiency, there is always a tradeoff but the most important thing is how the impact of that tradeoff can be minimized. The architecture proposed and implemented in this paper hosts the modules running the malware detection algorithms, which are discussed in the later sections. Any security system, which claims to identify a malware and protect the system in real-time with sophisticated techniques will require a lot of resources for doing so and ends up making the system more vulnerable than making it resilient. The algorithms and techniques that are proposed, implemented, and assessed in the later sections have the main objective to detect malware by a set of different techniques. To avoid burdening the host machines, which require protection against malware attacks, majority of the system is hosted on Amazons cloud platform.

The architecture present in figure 5, illustrate that when a client sends a request, it is sent to a queue, which is pulled by the detection engine to perform the analysis and the response is then sent by the detection engine to the client through the queue as well. To make decisions that whether a file is clean or malicious and that too accurately and instantly, the detection engine requires constant and rigorous training. The algorithms used to build this detection engine, discussed in the later section, are trained continuously by analyzing a large amount of clean and malicious files to understand the difference between the both. These files are stored in malware repository, which is hosted on Amazon Elastic File System (EFS) which makes it easier to manage and update remotely. The reports generated after the analysis are also stored in the same environment. Hosting the entire system on different modules of AWS not only makes it more scalable, it is extremely reliable and efficient.

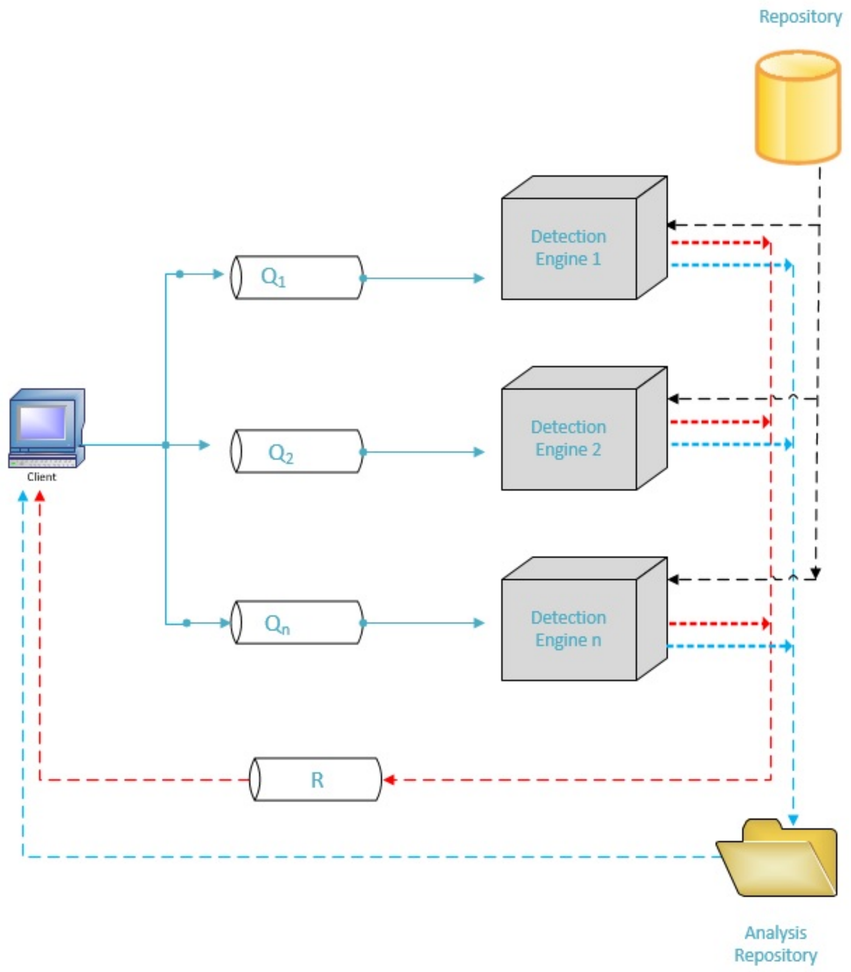


Figure 5: CloudIntell System Architecture

The architecture illustrates five main modules in the overall system, four out of them are hosted on the cloud. It can support multiple individual clients and many enterprise networks concurrently. The experiments performed in this research used single request and response queues and only one detection engine because of low number of requests. However, the architecture is designed and implemented in a way that it can dynamically enhance itself based on the higher number of requests or higher number of clients. This means that the same architecture can cater the security needs of a distributed enterprise network or multiple enterprise networks.

3.1. Queues

Figure 5 illustrate the functionality of both request and response queues in the overall architecture, both queues are responsive in nature and based on the number and frequency of requests from the client module. These queues are developed and configured using Amazons SQS (Simple Queue Service), which is a purpose-built service for message queues fully managed by Amazon and works flawlessly between different distributed applications and microservices. Amazons SQS has elastic capabilities that allow the queues to dynamically scale up or down based on the systems overall requirement. The client-server architecture of the system requires a platform, which guarantees message delivery between client and server without any delay, it also requires a platform that manages and delivers the messages based on the server availability to avoid message loss. Additionally, the system also requires identifying each message and avoid any message replication. All these requirements are fulfilled by SQS along with full-scale management by service provider to eliminate the expensive overhead of dealing with message-oriented middleware and its hosting.

3.2. Detection Engine

As presented in figure 5, the requests sent by the client are forwarded to the detection engine by the queues. The detection engine runs all the techniques and algorithms that are later discussed, there are multiple detection engines that are hosted on Amazons EC2 (Elastic Compute Cloud). Amazons EC2 provides computing facility in the cloud with elastic behavior, which is available as a webservice. Detection engine requires a hosting mechanism which is dynamically scalable and extremely reliable when it comes to cater the client requests. It also requires itself to be replicated if the number of clients and number of requests significantly increase without causing any delay to other clients and requests already in process. Moreover, detection engine is required to connect with other modules included in the system which are designed to be distributed. Amazons EC2 fulfills all the hosting requirements of the detection engine quite efficiently and doesnt require to be managed in contract with other available options, such as; dedicated external server or an inhouse server.

3.3. Repositories

The two repositories shown in the architectural diagram; repository and analysis repository that store files to be analyzed (both clean and malicious) and analysis reports respectively are also part of the server module. Like other server modules, both repositories are also required to be distributed and scalable. The repository stores both clean and malicious files, which are used to train and test the algorithms that are discussed in the later sections. The number of files in this repository vary from time to time and the ideal situation will be if the hosting mechanism can scale itself up and down based on the number of files and the number of analysis reports stored in it. Amazons EFS (Elastic File System) allows dynamic scalability along with an ease of connectivity with other distributed applications and service interfaces. Amazon EFS flawlessly connects with EC2 allowing the applications hosted on EC2 to be integrated with the filesystem in a larger distributed environment ensuring highest level of reliability at all times.

To protect the client from a malware attack, there is cloud-assisted lightweight agent installed on the client system, as proposed, implemented, and tested by [10]. This lightweight agent can detect malware to a certain extent by inheriting some of the intelligence from the cloud detection engine. When the client-based agent detects a file, it sends the file to the cloud engine and simultaneously checks for legitimacy of the tested file, if the client-based agent can classify it as malicious then it blocks the file and send the report to the cloud-based detection engine, if the client-based agent is not able to identify it as malicious then the detection engine sends the response based on a detailed yet rapid analysis. Usually, the cloud-based detection engine doesnt need to analyze the file in detail because of its rich algorithm and if there is a need for the analysis, it is done without any extra time. The architectural diagram shows a blue dotted line coming out of the detection engine and going to the client through the analysis repository, which shows that if a request comes from a client the detection engine first checks if that specific file or its variant is analyzed earlier by checking in the analysis repository. If it exist in the repository, then a response action is sent back to the client, which makes the overall process much efficient.

4. Data Collection

As stated earlier, the data for this research consisted of 150000 malicious files and 87000 benign executables of Windows PE format. The benign executables were retrieved from fresh installation of Windows 7, Windows 8, Windows 10, Windows Server 2008, and Windows Server 2012. The malicious files present in the malware repository were obtained from our industrial partner Nettitude, which was a combination of different malware types as presented in table 2.

Malware Type	Percentage
Trojan	65.82%
Adware	22.67%
Worm	8.66%
Downloader	1.21%
Virus	0.56%
Dropper	0.41%
Exploit	0.39%
Spyware	0.28%

Table 2: Malware Distribution in the Repository

The files, both benign and malicious, present in their respective repositories were used to train and test the algorithms.

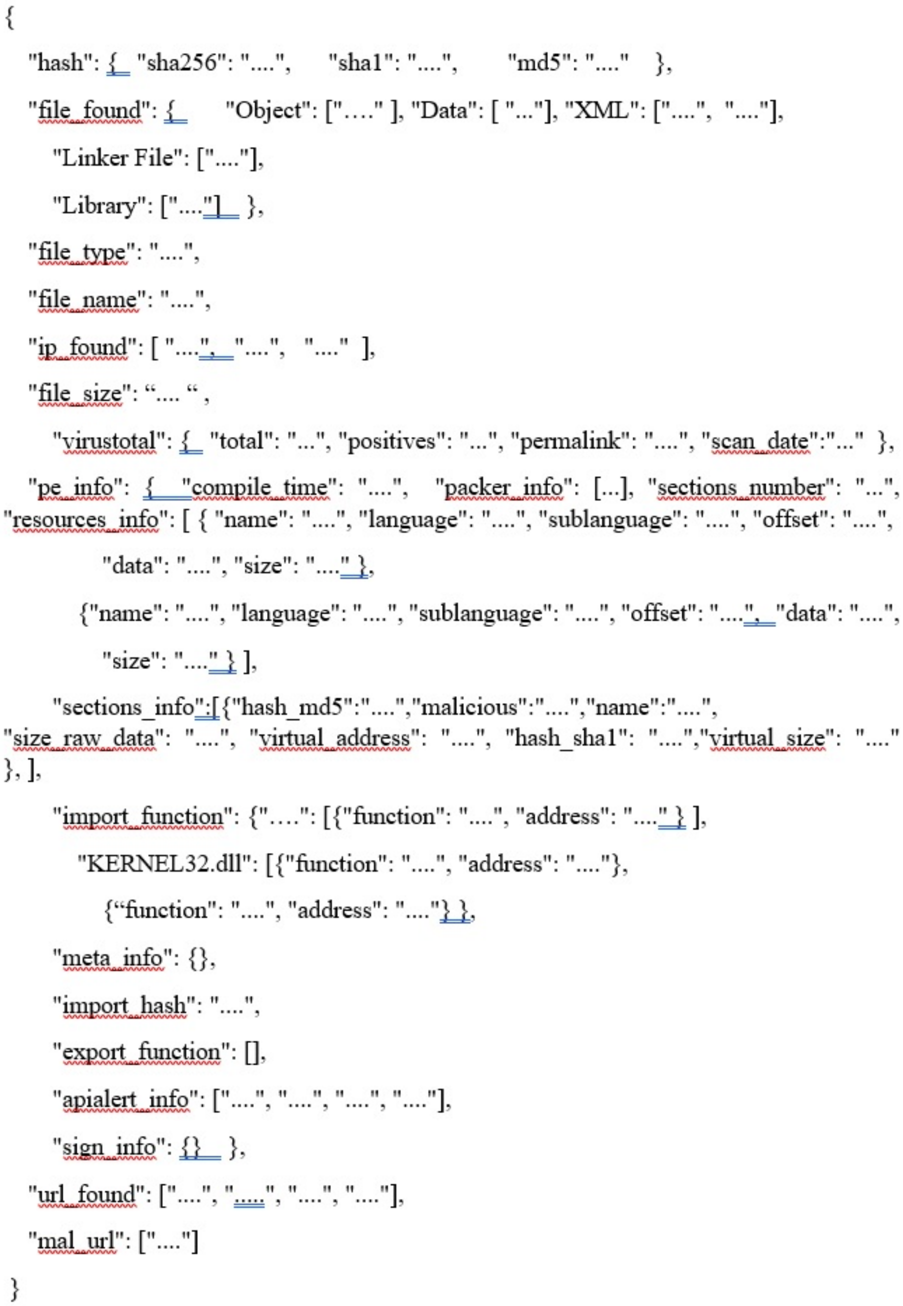
The distribution of both type of files is presented in table 3. Apart from the files in the repositories, the client-based agent captures files in real-time, which are only used to check for legitimacy and then stored in the respective repository. The files retrieved through the client-based agent are either captured through the browser extension or through process logs [10]. Either ways, these files also become part of the main repository.

File Type	Quantity
Benign	87000
Malicious	150000

Table 3: Distribution of Benign and Malicious Files

To extract relevant features, we developed a python-based tool that comprehensively analysis the file and generates a rich set of features. We incorporated our automated tool with another open source tool PEframe [19], which enhanced the overall functionality of the tool and personalized it to be flawlessly integrated with our cloud-based system. The system automatically generates JSON-based files after statically analyzing a malicious or benign file and each JSON file contains a rich set of decisive features, which are then used to training, testing, and detection purpose. The set of features generated by our tool, unlike our initial research [10], integrates the results of multiple antiviruses by incorporating private API of VirusTotal [20]. The use of VirusTotal API not only makes the JSON files richer, it also endorses some of the analysis results from our tool.

Figure 6 presents the features that are extracted by the analysis tool from every file it analyzes, it also presents the format in which the JSON file is populated with all the extracted features. The set of features in figure 6 start with the



```
{
   "hash": { "sha256": "....",     "sha1": "....",       "md5": "...."    },
   "file_found": {      "Object": ["…." ], "Data": [ "..."], "XML": ["....",  "...."],
      "Linker File": ["...."],
      "Library": ["...."]  },
   "file_type": "....",
   "file_name": "....",
   "ip_found": [ "....",  "....",   "...."  ],
   "file_size": “.... “ ,
      "virustotal": {  "total": "...", "positives": "...", "permalink": "....", "scan_date":"..."  },
   "pe_info":  {  "compile_time":  "....",   "packer_info":  [...],  "sections_number":  "...",
"resources_info": [ { "name": "....", "language": "....", "sublanguage": "....", "offset": "....",
            "data": "....", "size": "...." },
         {"name": "....", "language": "....", "sublanguage": "....", "offset": "....", "data": "....",
            "size": "...." } ],
      "sections_info":[{"hash_md5":"....","malicious":"....","name":"....",
"size_raw_data":  "....",  "virtual_address":  "....",  "hash_sha1":  "....","virtual_size":  "...."
}, ],
      "import_function": {"….": [{"function": "....", "address": "...." } ],
         "KERNEL32.dll": [{"function": "....", "address": "...."},
            {“function": "....", "address": "...."} },
      "meta_info": {},
      "import_hash": "....",
      "export_function": [],
      "apialert_info": ["....", "....", "....", "...."],
      "sign_info": {}  },
   "url_found": ["....", "....", "....", "...."],
   "mal_url": ["...."]
}
```

Figure 6: Feature Extraction (without values) and JSON Format

basic and most important feature of any file i.e. the hash. Our analysis tool extracts three hashes; SHA256, SHA1, and MD5, which are presented as three different elements of one set. The analysis tool also extracts the libraries used by the analyzed file, type of file, and any other files that are linked with the analyzed file. One of the most interesting things extracted by our analysis tool IP addresses that the said file is trying to access, which helps to determine if the file is trying to access any malicious IP address. Moreover, critical details including API calls, and URL calls are also extracted, which significantly enhance the understanding of the legitimacy of the file based on its calls to malicious or non-malicious external calls. Additionally, to get an external endorsement from 57 antiviruses, our tool incorporates the private API of VirusTotal.com and returns the decision based on the outcome of 57 antiviruses, which is later used in the process of decision making.

Along with the conventional signatures used by antiviruses, our tool generates some useful features which can make the detection more efficient for the system. The packer used by the malware author to pack the file is also extracted as a feature to identify the legitimacy of the file, as proposed and implemented by [15], pattern recognition can be applied to identify an entire family of malicious files by identifying the packers. We used a slight different technique by identifying the major classes of packers and how they are used to create variants of old malware by applying machine learning. The rich set of features or attributes present in a JSON file are used to train and test the algorithm. The most important part of the whole research is the set features which is used for training, testing, and detection. We finalized the most relevant and pivotal features, which make the system more efficient in identifying the true positives. Some of the features we extracted have the binary value and some have percentage or a non-binary value, which makes the decision based on this set more accurate.

5. Methodology

The approach taken in this research is combination of conventional and unique methodologies for malware detection. We combine detection techniques commonly used by antiviruses with state-of-the-art machine learning algorithms to make the system more robust and decisive against modern malicious attacks. The features extracted from benign and malicious files form a rich and diverse set of features, as discussed in the previous section. We used two different machine learning techniques and take full advantage of the rich set of features acquired through the analysis of malicious or benign files.

In the earlier research [10], we implemented ten antiviruses in a detection module and linked it with the static analysis module, both of these modules were in a serialized manner separately. In this research, we combine both the techniques and created one module that consumes lesser energy and generates a significant number of eloquent features and that too in a coherent format.

5.1. Classification Methodology

As mentioned earlier, apart from the conventional detection techniques, we adopted two different techniques from machine learning. We used the refined features extracted from the PE files to create training samples by considering each feature or attribute as Boolean, which is either exist True or not False.

We initially selected a small sample of malicious and benign files and applied the following learning techniques, which we implemented in MATLAB; SVM (Support Vector Machines), decision trees, we then applied boosting decision tree. The objective of our methodology is to detect malicious files, therefore, we refer to malicious files as positives class and benign as negatives class. The learning techniques mentioned above are discussed in the following sections.

5.1.1. SVM (Support Vector Machine)

Support vector machines, is a training algorithm which presents a decision boundary by maximizing the margin amongst training patters. The algorithm presented by [22], has performed in an optimal fashion in many conventional scenarios, along with some studies similar to ours [13, 14, 17]. SVM creates a linear classifier, therefore, vector of weight $\vec{w}$ is its concept description and a threshold or an intercept, b. To make the problem linearly separable, a kernel function is used by the SVMs for mapping training data into a higher-dimensioned space. To set $\vec{w}$ and b that hyperplanes margin is ideal, quadratic programming is used, which means that distance to the closest examples of negative and positive classes is maximum from the hyperplane. While running, if $\langle \vec{w}.\vec{x} \rangle - b > 0$, positive class is predicted and if vice versa negative class is selected by the method. However, for larger set of problems, quadratic programming can be complex and expensive, whereas, to train SVM efficiently, SMO (Sequential Minimal Optimization) is a much better algorithm [22], it computes the probability of positive and negative class during execution [23]. For performance, we used implementation proposed in [23] for computing each classs probability and then we used positive classs probability as the rating. We used the following linear SVM formula to predict the positive classes:

$$t(x) = \sum_{n=1}^{N} \omega_n K(x, x_n) + \omega_0 \qquad (2)$$

5.1.2. Decision Tree

A decision tree is decision support mechanism with nodes that represent attributes and the leaf nodes that represent the class labels. Branches of the tree that lead to children represent the values of the attribute. Values of the attributes and those attributes of an instance are used by the performance element to navigate in a tree starting from root and leading to leaves or an individual leaf. By choosing the attribute that perfectly separates the training samples into their appropriate classes, this is how a learning element generates a tree. Node, branches, and children are created for the attribute and the value of the attribute, the attribute is then eliminated from additional consideration, and the examples are distributed to the relevant child node. This process runs in a loop until the same class examples are stored in a node and then class label is stored. Many implementation of decision trees remove subtrees which are expected to perform inaccurately on test samples, which avoids the overtraining of the whole algorithm. We have used MATLAB decision tree implementation for training and testing.

5.1.3. Boosting

Boosting is used for combining multiple classifiers to enhance the performance as compare to individual classifiers [24]. It uses ensemble methods, which significantly increase the overall performance, which has been tested and endorsed by many studies [25, 26, 27, 28]. By repetitively learning from a weighted dataset of a model, it creates a set of weighted models by assessing, and revising the dataset based on the performance of the model. During execution of the method, to predict the highest weight class, it uses a set of models and their weights. We only applied boosting on decision tree implementation, as our initial experiments didnt show any significance of applying boosting on SVM. We used AdaBoost.M1 algorithms [24] implementation in MATLAB to boost decision tree.

5.2. Methodology Design

The machine learning techniques discussed in the previous section are incorporated with the analysis and feature extraction techniques to develop a comprehensive methodology, which can efficiently and accurately identify the malicious files. Figure 7 illustrates the complete methodology that is used for malware detection, this complete system is hosted on the cloud-based system discussed earlier and illustrated in Figure 5.

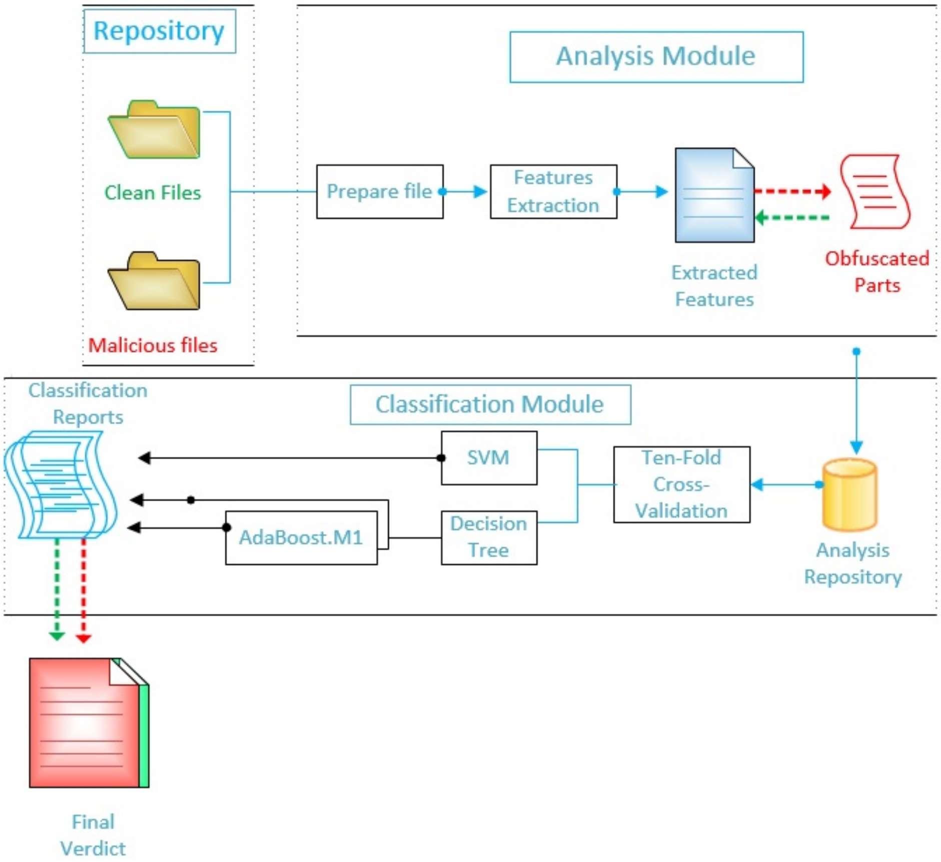
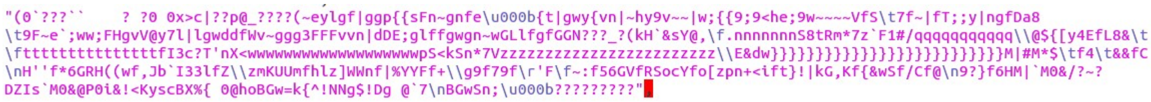


Figure 7: Proposed Methodology

We initially had benign and malicious files stored separately in the cloud repository, before applying any classification methodology every file has to go through the analysis module. The customized analysis module developed in this research first prepares the file by eliminating any apparent obfuscation, it is then thoroughly analyzed and all the possible features are extracted from the file. One of the significant features of the proposed methodology is that it is extremely effective on modern malware, which have the ability to mutate themselves while propagating in a network, which is mainly achieved by eliminating obfuscated parts from the extracted features and arrange the features in a format that is understandable by the classification module. Figure 8 presents the obfuscated part removed from the extracted features to make the features understandable for further classification.

The output of the analysis module is stored in the analysis repository. To perform the experiments we put all our benign and malicious files in the respective parts of the repository and our automated analysis tool extracted the features and stored the well-formatted features in the analysis repository. To initiate the classification experiments, the classification module pulled the features from the repository to apply the above mentioned machine learning techniques.



```
"(0`???``    ? ?0 0x>c|??p@_????(~eylgf|ggp{{sFn~gnfe\u000b{t|gwy{vn|~hy9v~~|w;{{9;9<he;9w~~~~VfS\t7f~|fT;;y|ngfDa8
\t9F~e`;ww;FHgvV@y7l|lgwddfWv~ggg3FFFvvn|dDE;glffgwgn~wGLlfgfGGN???_?(kH`&sY@,\f.nnnnnnnS8tRm*7z`F1#/qqqqqqqqqq\\@${[y4EfL8&\t
\ftttttttttttttttfI3c?T'nX<wwwwwwwwwwwwwwwwwwwpS<kSn*7Vzzzzzzzzzzzzzzzzzzzzzzzzz\\E&dw}}}}}}}}}}}}}}}}}}}}}}}}M|#M*$\tf4\t&&fC
\nH''f*6GRH((wf,Jb`I33lfZ\\zmKUUmfhlz]WWnf|%YYFf+\\g9f79f\r'F\f~:f56GVfRSocYfo[zpn+<ift}!|kG,Kf{&wSf/Cf@\n9?}f6HM|`M0&/?~?
DZIs`M0&@P0i&!<KyscBX%{ 0@hoBGw=k{^!NNg$!Dg @`7\nBGwSn;\u000b?????????"
```

Figure 8: Obfuscated Part of the Extracted Features

The methodology diagram illustrates that to produce a final verdict, the process went through ten-fold cross-validation to train and test the algorithms to evaluate the performance of the methodology and later based on this training it can detect individual malicious files while running in real-time.

6. Malware Detection Experiments

Previous sections present the details of architecture and methodology of the approach we have proposed in this research. In this section, we present the observations of multiple experiments performed to evaluate the methodology proposed. We performed four different experiments on a standard experimental design to help us evaluate the methodology from different aspects. In our first experiments, we used a smaller dataset of both benign and malicious executables and applied all the classification techniques. In the second experiments, we used a larger dataset to apply the classification techniques to monitor the enhancement in the detection rate. In the third experiment, we introduced obfuscated malicious files that are previously unknown and make the overall dataset much larger, which allowed us to observe the performance of the classification techniques against a much difficult dataset. The fourth experiment was performed on real-time data where we left the system running for more than two months.

6.1. Experimental Design

We have used stratified ten-fold cross-validation to evaluate the methodologies we are using. This is applied by distributing the files randomly into ten separate sets of same size. One set is selected for testing and the remaining nine sets are combined together to create a training set. We applied ten-fold cross-validation by conducting ten similar executions by using each of the ten partition as a testing set in each iteration. In each iteration, relevant features, discussed in the earlier section, were extracted from the training set and the classification techniques were applied on it. The outcome of the classification was used identify the files in the test set.

For the ROC (Receiver Operating Characteristic) analysis [26, 27] and to implement the ten-fold- cross-validation, we extracted the features through our analysis tool and imported in our machine learning build integrated with the MATLAB ROC code.

6.2. Experiment with Smaller Dataset

After extracting the features, all the parameters required to perform the experiments were achieved. We separated a set of 500 files from each category to conduct the initial experiment by starting with ten-fold cross-validation and then proposed classification techniques. The ROC curves of this experiment is presented in figure 8 and the areas under curves are presented in table 4. It can be observed from the table that applying boosting on Decision Tree enhance the outcome.

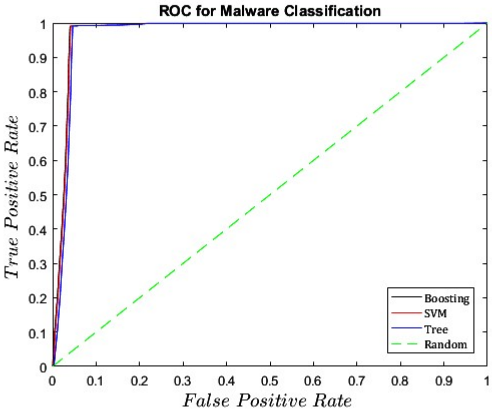


Figure 9: ROC Curves for Malware Classification from a Small Dataset

Method	AUC
Decision Tree	0.9708
SVM	0.9727
Boosting	0.9747

Table 4: Results of Applying Classification Techniques on Extracted Features of Smaller Dataset

The experiment performed on the small sample of clean and malicious files gave a satisfactory output considering the learning samples were just 500, which shows that the classification techniques proposed in this research have the potential to perform much better if they are well trained with higher number of samples.

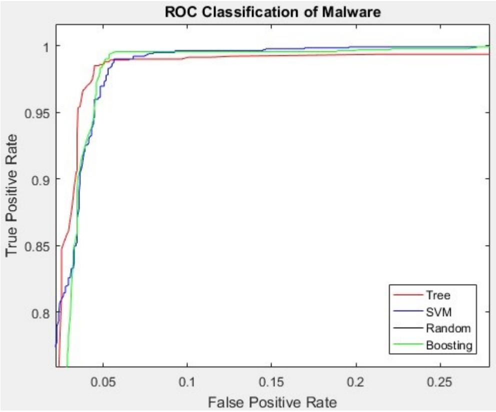


Figure 10: ROC Curves for Malware Classification from a Large Dataset

Method	AUC
Decision Tree	0.9775
SVM	0.9896
Boosting	0.9969

Table 5: Results of Applying Classification Techniques on Extracted Features of Large Dataset

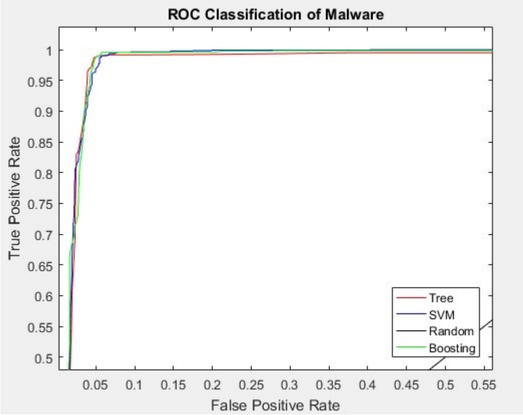


Figure 11: ROC Curves for Malware Classification from a Obfuscated Dataset

Method	AUC
Decision Tree	0.9740
SVM	0.9823
Boosting	0.9910

Table 6: Results of Applying Classification Techniques on Extracted Features of Obfuscated Dataset

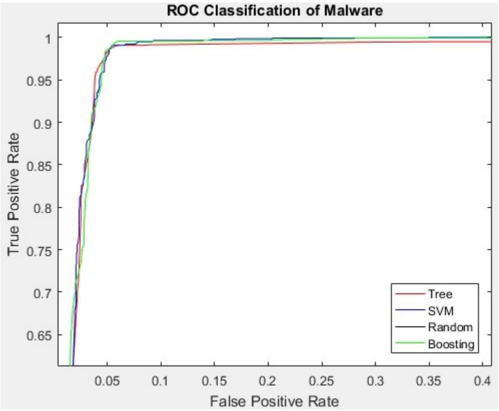


Figure 12: ROC Curves for Malware Classification from a Real-Time Detection

Method	AUC
Decision Tree	0.9765
SVM	0.9892
Boosting	0.9963

Table 7: Results of Applying Classification Techniques on Extracted Features from Real-Time Detection

7. Discussion

We performed four different experiments to evaluate our methodology and system architecture. The results presented in the previous section prove that the classification methodology proposed in this research prove the initial hy-

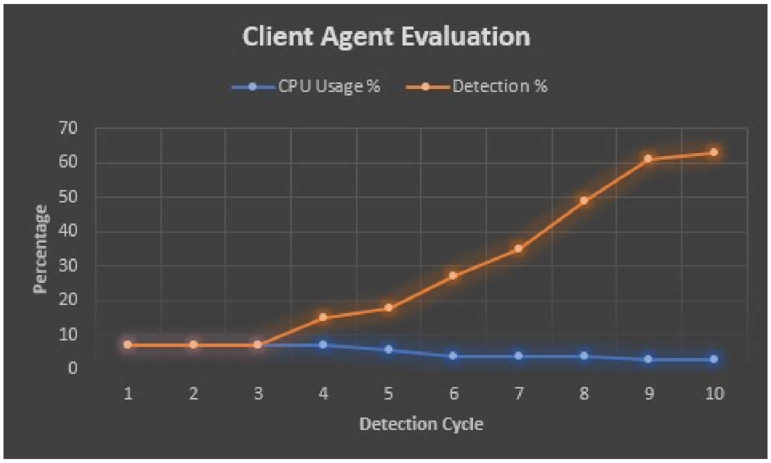


Figure 13: Performance Evaluation of Lightweight Client Agent

pothesis of enhanced accuracy in malware identification. The initial experiment tested the methodology with a smaller dataset to understand the effectiveness of the methodology. The results achieved from the experiments performed on the smaller dataset were quite satisfactory, as presented in figure 9 and table 4, but not better than the similar experiments performed by [12, 13], which achieved a better AUC as compare our approach. However, the reason our initial experiment lack the higher AUC was the feature set selected for training and testing, which required more testing. This was proved in the second experiment performed on a large dataset. The results of second experiment presented in figure 10 and table 5 show remarkable improvement over the previous experiment and also comparing to [13, 14]. SVM achieved a better rate than decision tree but the implementation of boosting on decision tree significantly enhanced the performance by producing a much higher detection rate. As discussed earlier, boosting can enhance the performance of unstable classifiers by decreasing their variance and bias but it can work inversely on stable classifiers [24], which is why we only applied boosting on decision tree and not on SVM.

Our experimental results prove that CloudIntells methodology can scale in performance on a larger set of files. The training and testing performed on larger dataset was extremely rigorous because of the feature set and techniques used, which also proved that modern obfuscated malware can also be identified with accuracy, as illustrated in figure 11 and table 6, presenting the result of applying techniques on obfuscated dataset. Evaluating classification methodologies based on machine learning against obfuscated and mutated dataset has not been performed by [13] and many studies [16, 15, 30, 31, 32]. This also shows the versatility of our approach that can be applied on multiple security threats and can identify not just known but it can also predict unknown threats.

To evaluate the methodology against live threats, we left the entire system running for more than two months. There were two objectives behind this experiment; one was to evaluate the system accuracy to detect live threats, second was to evaluate the energy efficiency of the lightweight local agent. The proposed methodology showed extremely good results outperforming [13] in their similar experiments of real-time detection of malware. The lightweight host agent also performed extremely well. Figure 13 presents the evaluation of the lightweight client agent. The evaluation process of this client agent involved ten cycles of malware detection and each cycle included 15000 files. The evaluation was performed on every cycle, which means that after every 15000 files scanned by the client agent we gathered the information about the number of files that were locally detected as malicious and the percentage of CPU resources used in that cycle. As illustrated in figure 13, in initial cycles the local detection rate is zero, which means that the client agent is unable to detect malicious files locally and detection engine supporting the detection mechanism and at the same time the client agent is importing the local cache from the cloud-based detection engine to enhance the local detection. Therefore, after three cycles the local detection starts to rise and simultaneously the CPU resource consumption of the client agent starts to go down and it can be seen at the tenth cycle, 60% of 15000 files are detected locally and the CPU resource consumption dropped to 3% only.

Moreover, the detection rate of our proposed methodology is better in many different than related work as discussed in this section but we were unable to compare the systems overall performance results with any previously available study. We were unable to find any study that combines different machine learning algorithms and deploy a comprehensive distributed architecture based on cloud to enhance the detection and reduce the client's CPU consumption.

The architecture of CloudIntell presented in figure 5 is capable of managing a large number of requests coming from multiple individual clients and enterprise networks. However, we didnt test the cloud-based architecture against a large number of clients or against a big network. The test we performed showed scalability but the scalability of the architecture is yet to be tested. Nevertheless, the experimental results do suggest that the system has real-life industrial application and can significantly enhance the detection mechanism with every iteration.

System architecture, methodology, and their performance have been discussed using different aspects, such as; detection rate of algorithms both individual and combine, detection rate of lightweight client agent using ten cycles, and CPU resource consumption of lightweight client agent during the evaluation cycles. Another aspect of evaluating the whole system would be performance evaluation on a local system instead of cloud. We didnt perform the overall system performance evaluation locally but without even testing we can consider some basic differences amongst the two approaches; 1) system requires dedicated storage of more than 300GB to work as a repository, 2) Detection engine requires at least 4 GB of RAM to run the combination of algorithm and the static analysis tool, 3) Detection engine also require dedicated bandwidth to interact with the external APIs for decision endorsements. Although, the cloud-based detection engine and other module dont always require the above-mentioned resources and rarely reach this threshold of resources but when required these resources should be available for the cloud-based module to execute its tasks. Therefore, the scalability feature of cloud benefits the discussed architecture by dynamically scaling up and down based on the requirements of the system and helps to avoid any unnecessary costs.

8. Concluding Remarks

Using machine learning techniques to identify malicious activity in a system or in a network have proved to be quite effective [16, 15, 30, 31, 32]. We focused on two main issues in domain of malware detection; a) accurately identifying a malware, and b) enhance energy efficiency of the detection mechanism. The methodology of CloudIntell used machine learning techniques to enhance the malware detection rate and a cloud-based architecture to support and host the methodology implementation. We used decision trees, SVM, and then applied boosting on decision trees to improve the performance of weak classifiers. We developed an automated feature extracting tool, which extracted the features from more than 200,000 files quite efficiently. The feature extraction tool also have the functionality of removing the obfuscated parts of the malicious file, which allows the relevant features to be extracted appropriately to apply the machine learning methodologies.

We designed multiple experiment to test our proposed methodology from different perspectives. We tested our techniques against a dataset of malicious and clean files and applied ten-fold cross-validation followed by above mentioned machine learning techniques for an unbiased set of experimental results. We used around 150000 malicious and 87000 benign files for training and testing.

Support Vector Machine performed better than decision tree but applying boosting on decision tree improved the performance by generating the best detector of 0.9969 area under the ROC curve. To evaluate the methodology against a much difficult dataset, we used a dataset of obfuscated malware, which used the training of previous experiment to detect the obfuscated malware. Boosting on decision tree generated 0.9910 area under the ROC curve. This not only proved the better performance against difficult dataset, it also suggest that previous training was enough to detect a different set of malware. We also evaluated the system with real-time data and generated 0.9963 area under the ROC curve.

We also tested our host based agent in the real-time experiment and observed an optimum level of energy efficiency and unsupervised malware detection ability with the help of local cache. However, we were unable to test the CloudIntells architectural design against multiple clients or enterprise network, which we plan to do in future. We plan to enhance the lightweight local agent to work in an enterprise environment and then evaluate the cloud-based architecture against large number of request. This will also evaluate the performance of the host agent along with the detection mechanism in a distributed and heterogeneous environment of an enterprise.

Acknowledgements

The authors would like to thank Dr. Jules Pagna Disso and Dr. Anitta Namanya of Nettitude Ltd. for their technical input during the course of this project. We thank VirusTotal.com for providing their personalized support and private API for detailed analysis reports. A special thanks to Mr. Ali Maina Bukar of Centre for Visual Computing, University of Bradford for providing his technical expertise in machine learning.

References

[1] B. Snell, Mobile Threat Report, Intel Security, 2016.

[2] G. AV-TEST, AV-TEST The Independent IT-Security Institute, 09-Jan-2017. [Online]. Available: https://www.av-test.org/en/statistics/malware/. [Accessed: 14-Jan-2017].

[3] D. Bruschi, M. Lorenzo, and M. Monga, Detecting self-mutating malware using control-flow graph matching, in Detection of Intrusions and Malware and Vulnerability Assessment, vol. 4064, 2006, pp. 129143.

[4] I. You and K. Yim, Malware Obfuscation Techniques: A Brief Survey, 2010, pp. 297300.

[5] M. Schiffman, A Brief History of Malware Obfuscation: Part 1 of 2, blogs@Cisco - Cisco Blogs, 15-Feb-2010. [Online]. Available: http://blogs.cisco.com/security/a-brief-history-of-malware-obfuscation-part-1-of-2/. [Accessed: 08-Sep-2014].

[6] V. Harrison and J. Pagliery, Nearly 1 million new malware threats released every day, CNNMoney, 14-Apr-2015. [Online]. Available: http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/index.html. [Accessed: 10-Dec-2016].

[7] D. Kozlov, J. Veijalainen, and Y. Ali, Security and Privacy Threats in IoT Architectures, in Proceedings of the 7th International Conference on Body Area Networks, ICST, Brussels, Belgium, Belgium, 2012, pp. 256262.

[8] P. Ducklin, Deutsche Telekom outage: Mirai botnet goes double-rogue, Naked Security, 29-Nov-2016.

[9] J. Leyden, Sh... IoT just got real: Mirai botnet attacks targeting multiple ISPs, 12-Feb-2016. [Online]. Available: http://www.theregister.co.uk/2016/12/02/broadband-mirai-takedown-analysis/. [Accessed: 10-Dec-2016]

[10] Q. K. A. Mirza, G. Mohi-Ud-Din, and I. Awan, A Cloud-Based Energy Efficient System for Enhancing the Detection and Prevention of Modern Malware, in 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), 2016, pp. 754761.

[11] BBC, One billion affected by Yahoo hack, BBC News, 15-Dec-2016.

[12] O. Santos, Identifying and Classifying Network Security Threats ¿ Network Visibility, 2008. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=791595. [Accessed: 15-Dec-2016].

[13] J. Z. Kolter and M. A. Maloof, Learning to Detect and Classify Malicious Executables in the Wild, J. Mach. Learn. Res., vol. 7, no. Dec, pp. 27212744, 2006.

[14] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers, Malware target recognition via static heuristics, Comput. Secur., vol. 31, no. 1, pp. 137147, Feb. 2012.

[15] L. Sun, S. Versteeg, S. Bozta, and T. Yann, Pattern Recognition Techniques for the Classification of Malware Packers, in Information Security and Privacy, 2010, pp. 370390.

[16] K. Rieck, P. Trinius, C. Willems, and T. Holz, Automatic analysis of malware behavior using machine learning, J. Comput. Secur., vol. 19, no. 4, pp. 639668, Jan. 2011.

[17] M. Kruczkowski and E. N. Szynkiewicz, Support Vector Machine for Malware Analysis and Classification, in 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014, vol. 2, pp. 415420.

[18] J. Z. Kolter and M. A. Maloof, Learning to Detect Malicious Executables in the Wild, in Proceedings of the Tenth ACM SIGKDD International Confer-

ence on Knowledge Discovery and Data Mining, New York, NY, USA, 2004, pp. 470478.

[19] G. Amato, guelfoweb/peframe, GitHub, 2016. [Online]. Available: https://github.com/guelfoweb/peframe. [Accessed: 29-Dec-2016].

[20] VirusTotal, VirusTotal - Free Online Virus, Malware and URL Scanner, 2016. [Online]. Available: https://www.virustotal.com/. [Accessed: 14-Jul-2016].

[21] B. E. Boser, I. M. Guyon, and V. N. Vapnik, A Training Algorithm for Optimal Margin Classifiers, in Proceedings of the Fifth Annual Workshop on Computational Learning Theory, New York, NY, USA, 1992, pp. 144152.

[22] J. Platt, Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Microsoft Res., Apr. 1998.

[23] J. C. Platt, Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods, in Advances in Large Margin Classifiers, 1999, pp. 6174.

[24] Y. Freund and R. E. Schapire, Experiments with a New Boosting Algorithm. 1996.

[25] M. V. Joshi, V. Kumar, and R. C. Agarwal, Evaluating boosting algorithms to classify rare classes: comparison and improvements, in Proceedings 2001 IEEE International Conference on Data Mining, 2001, pp. 257264.

[26] X. Carreras and L. Marquez, Boosting Trees for Anti-Spam Email Filtering, arXiv:cs/0109015, Sep. 2001.

[27] P. Viola and M. Jones, Rapid object detection using a boosted cascade of simple features, in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, vol. 1, p. I-511-I-518 vol.1.

[28] T. G. Dietterich, Ensemble Methods in Machine Learning, in Multiple Classifier Systems, 2000, pp. 115.

[29] J. A. Swets, Signal Detection Theory and ROC Analysis in Psychology and Diagnostics: Collected Papers. Psychology Press, 2014.

[30] J. A. Hanley and B. J. McNeil, The meaning and use of the area under a receiver operating characteristic (ROC) curve., Radiology, vol. 143, no. 1, pp. 2936, Apr. 1982.

[31] D. Gavrilu, M. Cimpoeu, D. Anton, and L. Ciortuz, Malware detection using machine learning, in 2009 International Multiconference on Computer Science and Information Technology, 2009, pp. 735741.

[32] R. J. Mangialardo and J. C. Duarte, Integrating Static and Dynamic Malware Analysis Using Machine Learning, IEEE Lat. Am. Trans., vol. 13, no. 9, pp. 30803087, Sep. 2015.

[33] H. V. Nath and B. M. Mehtre, Static Malware Analysis Using Machine Learning Methods, in Recent Trends in Computer Networks and Distributed Systems Security, 2014, pp. 440450.