Improved Anti-Virus Defense via Efficient Clustering Techniques

Subrata Acharya and Gayatri Joshi

Towson University, Towson, MD, USA

Summary
With the exponential growth in the Internet and its applications, threats and vulnerabilities have increased in similar proportions. The most common way of defense used by organizations for system security is an anti-virus tool. Unfortunately, these tools have not been effective enough in providing the required defense to the volume and sophisticated nature of today's cyber attacks. This paper proposes a proactive distributed and co-operative method to improve the defense provided by such anti-virus tools. Evaluation results show the strength of the proposed approach both on random and real-world data sets.

Key words: *Internet attacks, Anti-Virus, Worms, Malware, Reputation, Collaborative, Clustering.*

1. Introduction

In 1969 Advanced Research Projects Agency (ARPA) of DoD, later known as DARPA created ARPANET using packet switching, unlike circuit switching in phone lines to improve communications [1]. With the Internet in operation, the first notable computer virus, the Morris worm [2] was reported in 1988, creating significant damage to the distributed communication medium. Subsequently, the number of cyber attacks has increased exponentially from 2003 till date. The **C**omputer **E**mergency **R**esponse **T**eam **C**oordination **C**enter (CERT/CC) was started in December 1988 by DARPA, to monitor Internet threat incidents; Figure 1 illustrates the number of incidents reported by CERT from 1988 till date.

Current governmental agencies, defense institutions, health organizations, online trading sites and telecommunications companies are increasingly targeted by overlapping surges of Internet attacks [4]. Figure 2 shows the major areas being affected by cyber attacks. In recent times, computer viruses are being used in identity theft to steal personal information such as passwords and credit card information [5]. Much of today's malware is fueled by financially-motivated cyber criminals trying to gain access to valuable corporate, consumer and/or personal data [6]. Internet attacks have increased in volume and sophistication such that organizations find it extremely difficult to detect and mitigate the rate and scale of such vulnerabilities [4]. During the last few years, the number of vulnerabilities discovered in applications has far exceeded the number of vulnerabilities in the Operating Systems layer. Figure 3 shows the number of vulnerabilities from the network layer to the applications layer.

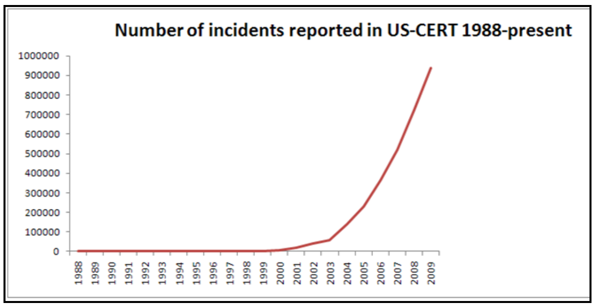

Fig. 1. CERT threat trends (1988 – present)

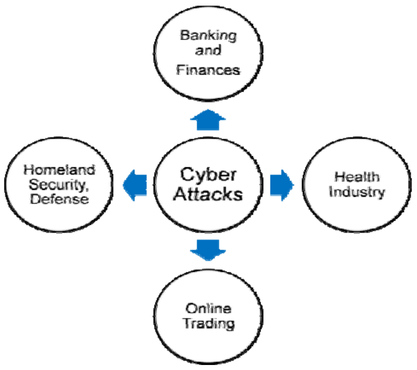

Fig. 2. Major targets for Internet Attacks

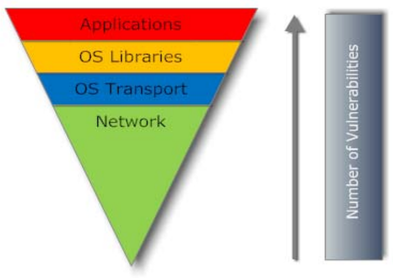

Fig. 3. Number of vulnerabilities in the OSI Layers

To address the volume, scale and diversity of such Internet attacks, present day computer systems need to have an up-to-date anti-virus tool running. This antivirus tool should

Manuscript received June 5, 2010
Manuscript revised June 20, 2010

be capable in determining the type of files (either malicious or good) in real time environments. To this effect such systems need some kind of predictive mechanism to be proactive in their detection and defense. In very recent research, "Reputation based systems" [7], play a pivotal role in providing a predictive mechanism in order to establish the nature (good or malicious) of the system files. In this paper we propose a "*Distributed Co-operative Detection Framework*", that the implements reputation model based and incorporate adaptive clustering approach to improve the classification of unknown files in the system.

The rest of the paper is structured as follows: Section 2 discusses the motivation and background of the problem. The current research trends are discussed in Section 3. Section 4 introduces the problem statement and presents the proposed "Distributed Co-operative Detection Framework". The evaluation details on both the emulated and random data-set are discussed in Section 5. Section 6 presents the conclusion and future research directions.

2. Motivation and Background

In this section we provide a brief overview of different types of malware to improve our understanding and aid us in designing countermeasures against such attacks. According to statistics twenty five million new strains of malware have been discovered in the year 2009, at a rate of one new strain per 0.79 seconds [8]. It is noteworthy to mention that United States Senate Security Operations Centre gets around 13.9 million cyber attacks each day [8].

Malware encompasses Viruses, Worms, Trojans and Root-kits. A virus is a piece of code that can copy itself and infect a computer. It is usually capable of causing irreparable damage to system files. It typically spreads by attaching itself to files (either data files or executable code). Virus spread requires the transmission of the infected file from one system to another. A worm does the same task of causing harm to files but without human assistance. Worms are capable of autonomous migration between systems through the network without the assistance of external software. Worms aggressively scan the network and attack hosts. Firstly the Worm scans the network for hosts with vulnerabilities and selects the target called *Reconnaissance*. They implement various scanning techniques such as hit list scanning or random scanning. When the target is identified the *attack component* launches the attack. The attacking node and the attacked node communicate using the *communications component*. The compromised nodes of the attack system can be issued commands using the *command component*. The worm scans for the vulnerabilities in the target system, exploits them and delivers its payload to the victim node so that it can now scan and propagate henceforth. The payload propagation can occur in numerous ways. The payload can be directly injected to the victim or it is passed from parent to child or there may be a central source to pass it every node. The worm payload might be compiled or interpreted. In addition to all these, it has an *intelligence component* that keeps track of the locations of all the nodes and their characteristics. This component provides information required to contact other worm nodes.

Worms usually select a target platform for better coverage. Microsoft Windows make up 90% of the client workstations surfing websites and 45% or more of the web servers on the Internet making them a popular choice of worms [2]. The more devastating worms have attacked Internet Information Server (IIS) web-servers. IIS has been a subject to scrutiny by the security community. As flaws have been found, exploits have been developed against them, some of these being incorporated into worms. Due to these attacks the number of incidents reported on CERT for Windows platform is much higher than those reported on Linux or UNIX platforms [9].UNIX based worms have a complicated design because of the variety of Unix platforms. The Apache worm was effective only against the FreeBSD systems even though many web servers run Apache on Linux and Sun Solaris systems [2]. The Slapper worm on the other hand was able to fingerprint several popular Linux distributions and launch a specified attack against them [2]. Worms saturate the network on which they reside.

A Trojan horse gains access to unauthorized services of a computer. It is embedded in some other form of code or software and it is non-replicating. Trojan code attempts to camouflage its presence to avoid detection. Once a Trojan horse has been installed, the attacker can remotely perform various operations. Rootkits gain administrative control over systems. They usually hide the utility systems for root access at a later date. These entry points are called *trapdoors*. All these malware attack the systems over the web to exploit the businesses and bring down services. When vulnerabilities in applications or services are discovered and the patches for those vulnerabilities are not yet out, there may be attacks that exploit these vulnerabilities. These attacks are known as *zero-day* attacks.

To protect the organizational network from these attacks, defense in depth mechanism is followed. Firewalls are used as choke points to drop unauthorized packet flow into the network. Intrusion Detection Systems (IDS) are used with Access Control Lists (ACL) to determine unusual activity. When an IDS identifies such an activity an alarm buzzes. Access controls have to be in place. Unnecessary services are disabled and unused ports plugged in. The operating system software is updated with security patches. Separation of duties or need to know factors play an important role in deciding whether a person should be granted access to a particular file. Every system in the network has an antivirus with updated signatures installed. All activities are logged for investigation in case of an incident, preferably on a once writable-append only remote location [11]. In spite of all these measures updated antivirus software is essential on every host. Otherwise that system can be compromised and used against the rest of the network. It can create a hole. If an employee who works from home or any other location uses an office laptop and that system is compromised, the security of the entire organizational network is at stake [11].

3. Current Technologies

In this section we describe in detail the common expectations from the tool, the steps taken by a typical antivirus tool to eradicate malware, all the technologies currently implemented by an anti virus tool and the drawbacks of the current technologies. Any sound design of an Anti-Virus tool should *Complement* application white listing technology for an effective defense-in-depth approach; combines traditional signature-based protection with unique behavioral analysis. It should *Prevent* known and unknown malicious threats (zero-day exploits) from gaining unauthorized access to systems and data, *Ensure* comprehensive clean-up, including rootkit removal and *Provide* fully automated operation, including new endpoint detection, signature updates, and easy-to-use web-based management console.

3.1. Anti-virus tool operation

Any anti-virus tool follows a set of steps to fulfill the above expectations. We enumerate these steps in the following. The understanding of these steps is important in accessing and improving the operation of the anti-virus software. *Assess*: This is the first step. This step uses signature-based scanning to identify viruses, worms, Trojans, key loggers, hijackers, root kits and other malicious software. Use behavioral analysis tools (including DNA Matching, Sandbox, and Exploit Detection) to assess suspicious code / applications [6]. *Remediate:* Prevent known malware and suspicious code from executing, and remove it from all network assets [6]. *Monitor*: Use customized triggers to generate alerts (delivered via e-mail, SNMP, SMS, Syslog and/or the operating system's event log) based on network-wide events (such as a spreading infection). Use Risk Level Indicator on web-based management console to understand overall network "health" and current event/client status of all endpoints [6]. *Report*: Use comprehensive, customizable reporting facility to cover entire network status and any incidents [6]. Figure 4 below depicts these steps.

3.2. Signature-Based scanning

Signature-Based scanning works on pattern matching. A dictionary of known fingerprints is used and run across a set of input. This dictionary typically contains a list of known bad signatures, such as malicious payloads or the file contents of a worm executable. This database of signatures is the key to the strength of the detection system.

Fig. 4 Steps of Anti-virus operation

There are three main types of signature analysis for worm detection. The first is the use of network payload signatures as used in network intrusion detection systems (NIDS). The detection methods used by NIDS engines perform an evaluation of packet contents received from the network, typically using passive capture techniques. This can include matching signatures based on application protocol analysis, or network characteristics.

Snort is a popular open-source NIDS package with some commercial support and a large user base. In case of the Code Red worm, a distinctive request is made to the target server that contained the exploit as well as the malicious executable. By examining the packets observed passively on the network, a detection system can identify Code Red worm activity. This signature looks for TCP packets to a list of Web servers on port 80. The payload of the packet is compared against the field. Upon matching, an alert is generated. This request is unique to the Code Red worm. The largest problem with this signature for Code Red worm is its size. Its signature is more than 100 bytes in length and must be fully matched against to successfully detect the worm's traffic. If this payload is fragmented due to network transmission sizes, the larger signature will not match the smaller payload in the fragments. A more reasonable approach would have been to focus on minimal unique identifier for the worm's traffic or a dozen or so bytes. For a signature that is too small, multiple false alarms will be raised.

The second type of signature matching is based on log file analysis. Application and system logs can contain information that can be used to fingerprint the behavior of a network worm. This can include attack contents, such as in Web server logs, or simple application errors issued when a worm probes a machine. This is a relatively simple approach but, when joined with other detection methods, provides a sophisticated detection framework. A small log file parsing script was developed during the spread of the Nimda worm. This script counted the number of requests that matched a basic signature for Nimda from each host. It looked for a pattern .exe in the Apache log file, which was a generic signature as the Nimda worm looked for the file root.exe on the target server. The script looked through the log files twice, the first time to generate a list of hosts that made requests that met these criteria, and the second time to count the number of requests made by the host and to record the last time the request was made. The underlying assumption was that these hosts were unlikely to make legitimate requests to the server, and that all requests were Nimda related.

The third type of signature detection is the most popular method, file signatures. File payloads of worms and their executables are typically monitored using host-level antivirus products. Examination of the contents of a file system is used to reveal the presence of a worm. Because most of the worms are binary executables and reside on the system's disk, looking for the worms signature on the file system makes sense. This method does not work for worms that are memory resident (like Code Red) or delete themselves after launching (like Morris worm). To examine the presence of these types of worms a virus detection tool that would scan systems memory would be required. Chkrootkit is a Linux and UNIX based file scanning tool. It is actually a suite of programs such as check_wtmpx, chklastlog, chkproc, chkwtmp and ifpromise and strings. It is possible to write such a tool for Windows but very difficult to maintain an updated list of malicious code and hence commercial malware detection tools are preferred.

Commercial antivirus products are the most popular methods used to detect worms. This is due to the popularity of their tools on Windows systems, making them numerous and widespread. The virus definition of an antivirus product contains a list of hexadecimal strings that is compared against the payload of files scanned on the system or in files being transferred, such as through electronic mail or via file server. The payloads of the files are compared against the virus definitions and the matches are noted with an alert. Some definition files are longer than others, with the length being dictated by the balance between a small enough file to scan efficiently and long enough to be a definitive match.

The biggest strength to signature-based scanning is the ease with which they can be developed and deployed. Once a worm (or any piece of malware) is captured or studied or even simply observed, only a brief analysis is needed to develop a signature. This analysis is performed to identify the characteristics that make the malicious software or traffic uniquely identifiable when compared against a backdrop of normal data. The features that are used in the monitor can be in the log file entries, the payload of files either on disk or in transit, or in the network traffic generated by the worm. The relative speed of signature-based detection systems is also another benefit of using them. Large number of optimized engines have been developed that can perform pattern matching efficiently, a requirement as communication volumes and the bandwidth of a typical network increase. These detection engines must keep up with this pace and react quickly. An additional benefit for signature-based detection methods is the ease of removal of the malicious content. For a mail or file server that is being used to distribute the worm, content screening immediately identifies the malicious payload and can quarantine the data. For a network-based intrusion detection system, reactive systems can be triggered to close a malicious connection or install a network filter on a router or firewall to block the compromised machine from continuing the worm's spread. Server level firewalls can also be configured dynamically by analysis engines once a malicious client has been identified from log file entries.

3.3. Anomaly-Based scanning

Anomaly-based antivirus tools determine normal behavior. Thus, any variation from the normal profile would be considered suspicious (anomalous). For example, normally a program, when executed, does not create any files. Then, all of a sudden, the program moves a file into one of the operating system's folders. That action would immediately be flagged by this type of antivirus software. Anomaly-based scanning typically works on predictive analysis. The tool learns normal traffic and predicts the future normal traffic. Figure 3 denotes a typical anomaly detection system. The tool has to learn the normal flow and develop a profile. It compares this traffic with the incoming packets. There are three main types of anomaly-based scanning approaches. The first type is called as Trend analyzers. Each entity (user or system) has its trend in communicating or generating network traffic. Email server generates a lot of email traffic or user network generates a lot of HTTP traffic and by identifying such trends it is possible to observe a trend in general network traffic coming through one point in the network. The second type is known as *Packet-analyzers*. Many packet analyzers try to verify if network traffic complies with RFC standards or generally accepted implementations. By detecting packets not complying with protocol standard or communication trend they raise the alert. One good example is *"arpwatch"*, which monitors ARP traffic and when detecting a change in MAC <-> IP relation it alerts the administrator. General communication standard of ARP traffic is that IP address doesn't change the MAC address in a static network (although there are some exceptions). Another good example is state-ful packet inspection, where state of communication is being monitored and in case of any deviation from this state alert would be raised (or the packet dropped). Based on the response period for monitoring and mitigation packet analyzers can be further divided into: *Long Term and Short Term*. **Long Term or** *Passive detection* uses scanning to detect derivations from the program's normal profile. **Short Term or Active detection** Involves executing a questionable program within a controlled environment such as a sandbox or virtual machine and then observing the behavior of the program. If the program meets certain negative criteria, it will be flagged as suspicious.

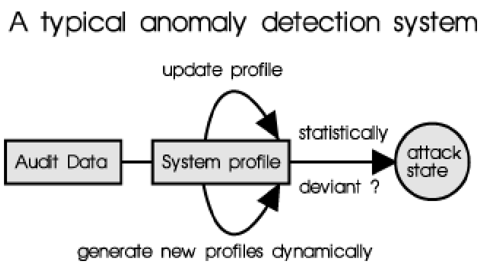

Fig. 5. A Typical Anomaly detection system

The third type is called *statistical analyzer*. Each network has its key identifications either qualitative or quantitative. By monitoring these identifications an IDS is able to detect anomalous traffic and report it. For example it is very suspicious to see increase of ICMP traffic from 1KB/s to 10MB/s or one IP address sending SYN packets to every port. *Threatfire Zero-Day Malware Protection* is an example of anomaly-based malware detection software.

3.4. Behavioral monitoring

Behavioral monitoring works on suspicion and *heuristics*. In the suspicious behavioral monitoring, the antivirus tool does not try to identify known viruses but monitors the behavior of all programs. If a program tries to write data on an executable file it flags the user. This approach, as against the signature based approach was designed to take care of new brand viruses whose signatures do not exist in dictionaries. However, with the advent of many non-malicious programs writing on executable files the false positive ratio increased. The heuristic approach is implemented either by *file analysis* or *file emulation.*

File analysis is the process by which antivirus software will analyze the instructions of a program. Based on the instructions, the software can determine whether or not the program is malicious. For example, if the file contains instructions to delete important system files, the file might be flagged as a virus. While this method is useful for identifying new viruses and variants, it can trigger many false alarms. In *File Emulation*, the target file is run in a virtual system environment, separate from the real system environment. The antivirus software would then log what actions the file takes in the virtual environment. If the actions are found to be damaging, the file will be marked a virus. But again, this method can trigger false alarms [12].

3.5. Hybrid scanning

Some tools implement *Hybrid* techniques to detect malware. Anomaly-based and heuristics or signature-based and behavior are some combinations used in Hybrid technology.

3.6. White and Black listing approach

Antivirus software uses file scanning, behavioral monitoring and hybrid methods for malware detection. They usually rely on the signature database. The signatures are either pushed into the client software or pulled by the client form the server. These updated signatures are based on the black and white listing of files. All valid files are stored in a hierarchy of distributed co-operating servers for example, applications such as Internet Explorer, Firefox, MS office and Adobe.. This is called a white list. Similarly, all known bad files are stored in another similar hierarchy and is called as a black list. When a file user is trying to execute exists in the black list it is blocked. If it is in the white list the antivirus allows the execution. There are a large number of unknown files [13]. Figure 3 depicts the white listing and black listing approach. The good files or valid applications are stored as hashes for faster processing. Prevalence is a measure to determine the existence and usage of the file over the Internet. This value is usually high for good files and low for bad files. This might however, need to be checked in case some files which were good earlier got infected. For our implementation we assume that good files are used at a very high rate as compared to the bad or infected files.

3.7. Drawbacks of current technologies

The biggest drawback to signature-based detection methods is that they are reactionary; they rarely can be used to detect a new worm. Only after an attack is known can it be fingerprinted and made into a signature for use by sensor. Only if the attack used by the worm is recycled from a known attack can it be used to proactively detect a worm. Some meta-signature detection methods, such as protocol analyzers and related tools that understand protocol parameters, can be used to detect a worm early on. However, there are uncommon in large, coordinated NIDS deployments at this time.

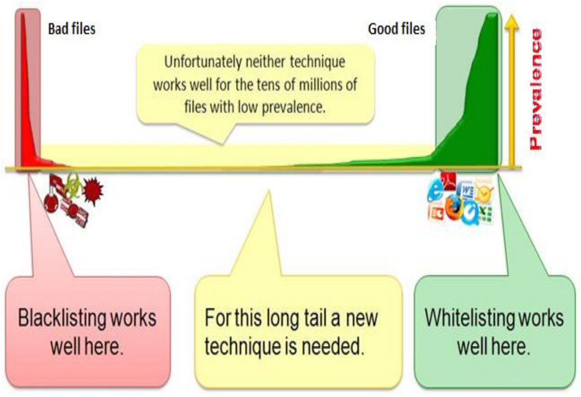
Bad files
Good files
Prevalence
Unfortunately neither technique works well for the tens of millions of files with low prevalence.
Blacklisting works well here.
For this long tail a new technique is needed.
Whitelisting works well here.

Fig. 6. Black and white listing

The second drawback to signature-based detection methods is that they don’t scale well to large operations. These include networks such as an enterprise or campus networks with thousand users. Desktop-based remedies are difficult to maintain actively, though many centralized management tools have been developed to overcome this obstacle. However, the volume and distributed nature of the problem makes the issue of scale a difficult challenge to adequately address.

The next major difficulty in a successful deployment of signature-based methods is that it is hard to keep up with variants of worms and viruses. Variations inevitably appear that can evade signature-based detection methods on all levels. Furthermore, when polymorphic techniques are introduced in the worms, the challenge raises significantly, making the reliable detection of worms much more difficult. Signature-based detection methods are only reactionary and always lag behind the introduction of the worm. Network-based signature detection suffers from a number of weaknesses, including payload fragmentation and forgery. Behavioral monitoring and heuristics have resulted in high false positives. Worms and Trojans implement all techniques to remain stealth. Sometimes attack the antivirus tool or block it. Polymorphic worms modify the encoding of the packet data. Black listing and white listing have a long tail of unknown files.

As good as this sounds, anomaly-based malware detection has shortcomings. False positives are more common with this type of detection, simply because of the complexity of modern-day programs. A small window will result in false positives while a large window will result in irrelevant data as well as increase the chance of false negatives. Techniques used by antivirus tools currently are static, non-scalable and has a very large number of unknown files. We need something else more efficient to find malicious activity well before the damage is caused. We can achieve that if we know about every file on the Internet. If we do not know about these files but know others opinions about them, we can still have some insight about them. We intend to use the knowledge of anybody in the world who might know about that particular file. If it is going to do something bad to our system, that can be avoided.

4. Framework and Methodology

We have seen the current technologies and their drawbacks and we realize that knowing about the files will help in decision making for the antivirus tool. Hence, we will focus on the white and black listing approach. The more information we have about a particular file the more proactive our anti-virus tool gets. First, we have to know about as much files over the internet as we can. We need a multi-level detection approach similar to the defense in depth implementation.

Our goal is to design a framework to classify unclassified files using a distributed co-operative detection method and strengthen the white and black listing approach to improve the anti-virus tool operation.

4.1. Distributed Co-operative Detection Framework

We address the problem of unknowns using the wisdom of crowds. The hierarchical framework of distributed co-operative systems is used to maintain the data and provide intelligence to the antivirus software. This framework is depicted in Figure 4. An Autonomous System (AS) is the system that has agreed to contribute to the reputation based approach. These play a vital role in constructing the model over time.

This architecture collects data from the AS, processes it, generates reputation for files and stores it. This reputation is then distributed to the network in the form of updates for the antivirus tool. Also, if the client application does not know about any particular file it refers to the hierarchy for decision. It is distributed and co-operative. This helps the processing of data and better utilization of the resources. We follow a cluster based approach in the framework. The workload is distributed amongst several machines over the network. The processing is done by every system and knowledge is shared. The distributed framework provides high availability. Since the framework is co-operative it is reliable to respond and there is no single point of failure.

Cluster based approach is useful in real time processing and faster response time for the AS from the co-operative detection framework. Real-time scanning scans files each time you or your computer access them. When virus protection detects an infected file, it tries to clean or remove the infection. If a file cannot be cleaned or removed, an alert prompts you to take further action. It is also important for scalability of the algorithm. A vast user pool plays an important role. We depict the system architecture in Fig.5.

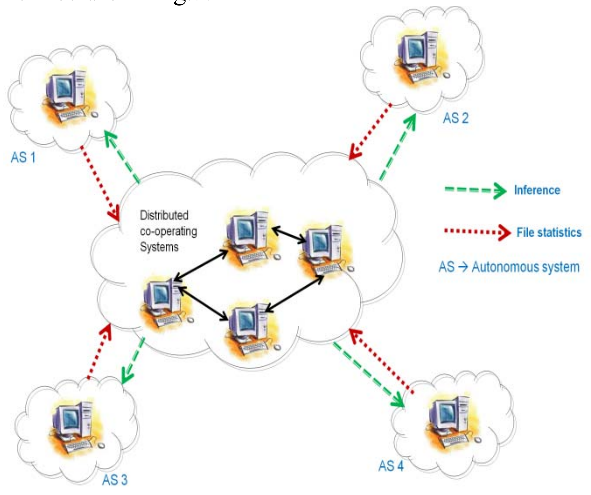

Fig. 4. Co-operative detection framework

It consists of contributing users, submission server and reputations server. There are several other servers that store white list and blacklist data but we will focus on this framework. One must note that all these servers form the detective framework. This architecture is an integral part of the co-operative framework. A limitation of this framework is the scenario of a massive targeted attack on an Autonomous System by Zombies, they might be able to manipulate its domain value.

Data collection:
Users of Autonomous systems (AS) opt in to contribute data. Co-operative framework starts collecting data. This includes several details about the file such as when was this file first seen on the Internet, its last access time and on how many machines it currently exists. This data has ranking for each file by every user. This rank is assigned automatically user does not have to "rank" them as such. This data is stored in the submission server and it is anonymous. Every AS is associated with a domain. This domain value depends on the trust factor about that system.

Aggregation:
Now the collected data has several ranks for any particular type of file. So we aggregate all these rank to generate one single value per file. Domain $=\sum X_i / m$ where Xi it the value allotted to the file by i_{th} AS and m is the number of machines ranking that file. For example in Fig. 4 AS 1 would be domain 1, AS 2 would be domain 2 and so on. This step gives us the number of copies of that file over the web. We now have the domain and the number of copies. The collected data also has age and last access time.

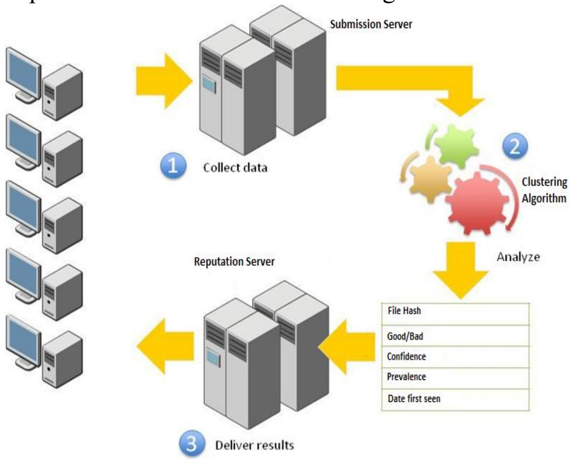

Fig. 5 System architecture

Prevalence calculation:
Prevalence value is calculated for each file. Prevalence for a particular file is defined as follows: Prevalence $=\sum N_i$, $N_i = \alpha$. Domain + β. Age + γ. Number of copies of that file over the Internet + δ. last access time. Where α, β, γ and δ are weights for that attribute. These weights are calculated experimentally. If some AS is sending false data to skew the prediction process, its domain value will decrease over time and the adverse effects on the reputation score will be nullified, meaning we no longer trust that AS. This calculation includes factors such as "the rating of an AS for a particular file as against its average rating for all the files" and "rating of an AS for a particular file in comparison with the ratings of other AS's for that particular file".

***Clustering*:**
We run the adaptive clustering algorithm on the prevalence values. We will see this algorithm in detail in the next section. This results in two different clusters. These two clusters help to predict if a file is good or bad.

Analysis and Inference:
Since we already have known good and bad files, we use their placement in clusters as reference points and infer the nature of an unknown file existing in the same cluster. We

check for the accuracy of this algorithm and determine the number of false positives in the outcome.

4.2. Adaptive Clustering Approach

Our focus will be on step 2 of Figure 5, where we try to convert the collected data into knowledge. We add the filtering algorithm [14, 15, 16, 17, 18, 19, 20, and 21] based on k-means clustering to this step. Basically, to come to a single prevalence value the system has to scan through numerous ratings (one per user). This can be a very tedious process leading to a slow response. The adaptive algorithm gives fast and efficient output to avoid this situation. It can handle very large datasets and gives same results as scanning the entire dataset.

```
Compute the $K_d$ tree for the given data points.
U= node, C=cell

Calculate weighted centroid. The weight is a vector sum of all associated points.

Select initial set of k centers

For each of the k centers,

        Compute the centroid of the set of data points for which the center is closest.
        A candidate set candidate set z* in Z, z* closest to the midpoint of C is chosen from the center points. These points might serve as the nearest neighbor for some point lying within as associated cell.

Prune data points not closer to the centroid

If the candidate is closer
        Replace centroid by the candidate
```

TABLE 1: ADAPTIVE CLUSTERING PSEUDO CODE

The adaptive clustering algorithm takes the prevalence values (one per file) as its input and gives two clusters as its output. These two clusters are dense enough to be recognized as good and bad clusters. The known good files and known bad signatures are used here to locate and identify the files in clusters. This algorithm randomly selects initial k centers, called centroids. For each of these k centers, there is a set of candidate points maintained. Each candidate has a set of associated points. The distance of these associated points is checked with the candidates. If it is closer to some other candidate, it changes its association. These associated points add weights to the candidates. Then the distance if the associated point is checked against the centroid or the k_{th} center. If the associated points are still closer to the candidate, the candidate becomes the new centroid. Table I gives the pseudo code for the algorithm. This way we just have to compare the randomly chosen candidates and not the entire stream of points and we still get the same results.

5. Evaluation

In this section we evaluate this algorithm on two different datasets for accuracy and false positives and state the test results. The evaluation was done on two datasets, an emulated dataset and a random dataset. The random dataset of 2000 files was used for evaluation. A random number generator was used to generate the prevalence values for these 2000 files. Testing on random dataset is important to know the accuracy of the algorithm. Also, because the random dataset comprises of only relevant fields and would give more relevant results. This would also work as a baseline for the data collection process to determine if some of the fields need to be dropped from the data collection phase. The emulated and random datasets were tested for multiple iterations.

Both the datasets were tested for classification accuracy and the number of false positives. Classification accuracy is determined by the improvement in the number of classified files. False positives are the good files wrongly classified as bad files. We also check if there are any false negatives i.e. bad files misclassified as good files.

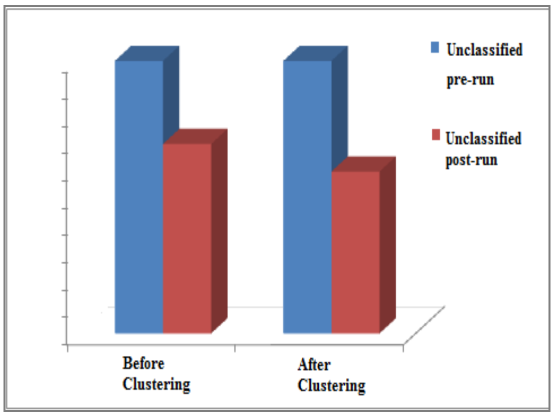

Fig. 6 Emulated dataset

The emulated dataset of unclassified files was tested on the algorithm without clustering. At the end it still had 69.5% unclassified files. The same dataset was tested on the algorithm with clustering. This time the unclassified files were 57.5%. This is a 12% classification improvement (Fig. 6). The Y-axis in the Figure 6 shows the total number of files.

A similar test on random dataset showed a 14% increase in the classified files. The result is depicted in Figure 7. The Y-axis on this graph shows the total number of files, in our case 2000. The test for accuracy was conducted on a random dataset of 2000 files. The test was conducted without incorporating the clustering algorithm and the after including the adaptive clustering module. The clustering algorithm increased the known good files by 13%, known bad files by 5% and the total increase in classification was 18% (Fig. 8). The Y-axis of this graph in Figure 8 is percentage.

A dataset of 300 files of which 225 were known good and 75 known bad was tested. All bad files were clustered together giving no false negatives. However, 7 known good files were clustered as bad giving a false positive rate of 3.1% (Fig. 9). The Y-axis if Figure 9 is number of files.

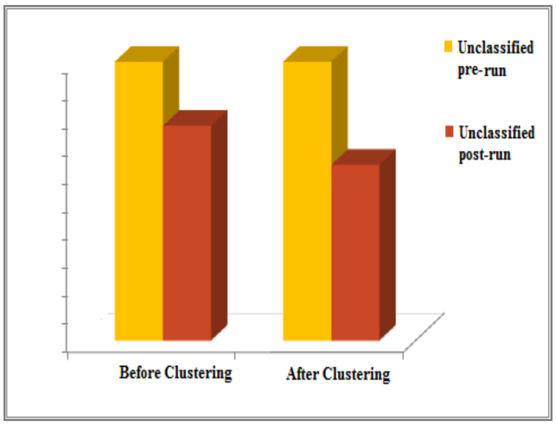

Fig. 7 Random dataset

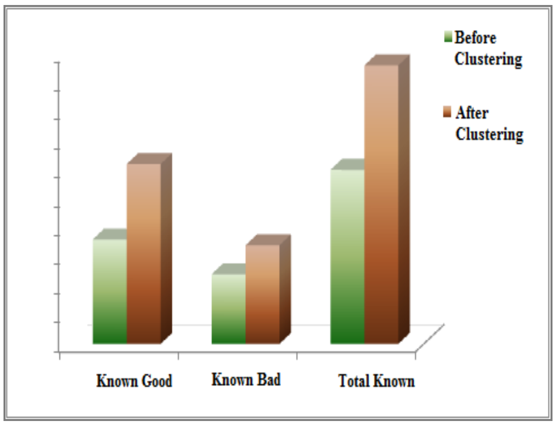

Fig. 8 Accuracy – random dataset

Based on the results above, we infer that the algorithm improves the classification rate by 12% on the emulated dataset and by14% on a random dataset with a few false positives and no false negatives. The addition of clustering helps in identifying more files as good or bad. In the end there are still a lot of files that do not fit in any of these clusters. We still do not know about these outliers. Hence, we could just minimize the gap to some extent and not to the totality.

A limitation of this framework is that it builds the reputation over time and any event may take some time to "sink in" to the system or propagate to the Autonomous System (AS). For example when an AS is compromised the weight reduction (α) would take some time. Also, since it is a self learning system no AS would remain black listed forever. This propagation time may raise the risk level of the AS. However, this can be mitigated by using it in collaboration with other technologies. It works the best when used to add another layer of protection to the tool. Building a reputation without enough data would be a challenge.

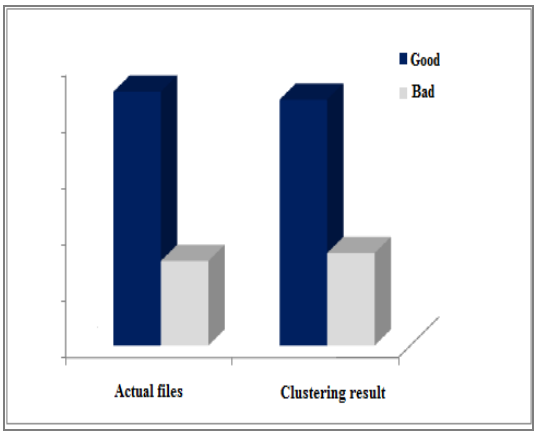

Fig. 9 False positives – random dataset

5. Conclusions and Future Research Directions

In this paper we discussed the different types of malware and their affects on businesses. We described the current technologies in use by antivirus tools and the white and black listing approach. We explained the framework architecture used for implementing the reputation based systems and the methodology to generate reputation for files. We then described the adaptive clustering algorithm based on modified k-means clustering. We evaluated it against the random and emulated dataset and stated the results.

Our results show increase in the known good and bad files, a total increase of 18% in the classification and a false positive rate of 3.1%. This raise in classification rate would enhance the decision making of the detective framework and give a better support to the users of AS. Hence our algorithm is an improvement to the current scenario. When used in collaboration with other technologies such as behavioral matching and heuristics it is bound to prove more effective by adding another protective layer.

Future research includes integration of the algorithm in antivirus tools for use in real world. Once integrated, the response time would need to be addressed with respect to the co-operative detection framework. It is obvious that the clustering algorithm would reduce the processing time as it just scans the representative instead of the whole stream. However, it is important to test the required time and the time required for the framework to respond to the AS in real time. Other than that, the scalability of the framework is also a subject to research. Another aspect of future research would be to test it on other real world datasets, i.e. virus data and defense datasets and analyze its behavior and test its performance. Our framework currently does not address the problem of Zombies. If a bot-net has compromised victims and these victims or zombies are trying to manipulate the reputation of any AS, the co-operative detective framework currently would not be able to address this issue. It assumes all coming from an AS data is good data.

References

[1]http://courses.ischool.berkeley.edu/i103/f07/slides/HofI 12- 10Internet.pdf

[2] Defense and detection strategies against Internet worms, Jose Nazario

[3] http://www.webopedia.com/TERM/C/CERTCC.html

[4] http://www.sans.org/top-cyber-security-risks/

[5]http://security.comcast.net/get-smart/security-trends/spotlight-virus-scan.aspx

[6]http://www.patchmanage.com/Lumension-Antivirus.asp

[7] http://www.symantec.com/security_response/index.jsp

[8] http://capec.mitre.org/data/graphs/1000.html

[9]http://kb.bitdefender.com/files/KnowledgeBase/file/security_report_windows_vs_linux.pdf

[10]http://www.defensesystems.com/Articles/2010/04/26/Digital-Conflict-Cyber-Defense.aspx

[11]http://www.easynetlive.info/behavior-monitoring.html

[12] www.us-cert.org

[13] www.Autonlab.org

[14] An efficient k-means clustering algorithm: analysis and implementation; Kanungo, Netanayahu et.al.

[15] Efficient clustering of High-Dimesional Datasets with Application to reference matching, Andrew mcCallum, Kamal Nigam, Lyle H. Ungar.

[16] An efficient clustering algorithm for Market Basket Data based on small large ratios, Ching-Huang Yun, Kun-Ta Chuang and Ming-Syan Chen.

[17] CURE: An efficient clustering algorithm for large databases, Sudipto Guha, Rajeev Rastogi and Kyuseok ShimAn experimental study of diversity with off-the-shelf Antivirus engines.

[18] An experimental study of diversity with off-the-shelf Antivirus engines, I. Gashi, V. Stankovic, C.Leita, O. Thonnard, Centre for Software Reliability, City Univ. London, London, UK Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium, July 2009.

[19] Eigen trust algorithm for reputation management in P2P networks, Sepandar Kamvar, Mario Schlosser, Hector Garcia-Molina.

[20] A computational model of trust and reputation, Lik Mui, Mojdeh Mohtashemi, Ari Halberstadt.

[21] http://cisecurity.org/en-us

[22] http://www.bsa.org/country.aspx?sc_lang=en

[23] http://cee.mitre.org/

[24] http://www.windowsecurity.com/whitepapers/

[25] http://capec.mitre.org/data/graphs/1000.html

[26] http://www.owasp.org

[27] http://www.first.org

[28]http://www.security-gurus.de/papers/anomaly_rules_def.pdf

Dr. Subrata Acharya is an Assistant Professor is the Department of Computer and Information Sciences at Towson University, Towson, MD. Her research interests are in Computer and Network Security, Distributed Systems and Real-Time Embedded Systems. Dr. Acharya received her Ph.D. from University of Pittsburgh and her M.S. from Texas A&M University.

Gayatri Joshi is a graduate student at Towson University pursuing a security track. She did her Bachelor's and Master's in Computer Science from Mumbai University in 2004 and 2006 respectively.