RAVE: Replicated AntiVirus Engine

Carlos Silva[1] Paulo Sousa[2] Paulo Veríssimo[2]
[1]Portugal Telecom
[2]LaSIGE, Faculty of Sciences, University of Lisbon
c.miguel.silva@telecom.pt, {pjsousa,pjv}@di.fc.ul.pt

***Abstract*—Antivirus is a fundamental presence in every computer infrastructure nowadays. The exponential growth of Internet usage with increasing higher bandwidth led to situations where virus (as well as worms and other type of malicious content) had constant outbreaks with impressive amounts of infected computers across the entire world. Email has been the preferred choice for several of these malicious content outbreaks. This paper describes the design, implementation, and evaluation of RAVE, a Replicated AntiVirus Engine for email infrastructures. Based on fault/intrusion tolerance concepts, this system allows to increase the detection capability of anti-malware solutions for email infrastructures by having different engines working in parallel, allowing at the same time arbitrary faults in a predefined number of replicas.**

***Index Terms*—fault-tolerance, security, antivirus, email, replication, arbitrary faults.**

I. Introduction

ANTIVIRUS is a fundamental presence in every computer infrastructure nowadays. Ever since people started to share files and using network services, viruses, worms and other malicious contents have become a growing presence in computers. The exponential growth of Internet usage with increasing higher bandwidth led to situations where virus (as well as worms and other type of malicious content) had constant outbreaks with impressive amounts of infected computers across the entire world. Email has been the preferred choice for several of these malicious content outbreaks, with various reported situations. To address the new threats, new security solutions were developed under the "umbrella" of antimalware products, with several detection engines to identify different types of threats, i.e., virus, worms, trojans, spam, phishing, spyware, adware. Moreover, the growing complexity of these products led to an increase in the number of vulnerabilities that can be later explored.

This paper describes the design, implementation, and evaluation of RAVE, a Replicated AntiVirus Engine for email infrastructures. Based on fault/intrusion tolerance concepts, this system allows to increase the detection capability of anti-malware (e.g., virus, spam, spyware, phishing, adware) solutions for email infrastructures by having different engines working in parallel, allowing arbitrary faults in a predefined number of replicas. By having a system with several replicas, each running a different antivirus engine (and, if possible, a different operating system), we obtain a system that offers a very high detection efficiency with virtually no downtime (even during antivirus' updates) while allowing the arbitrary failure of a predefined number of replicas, even if these failures are provoked by a malicious intruder that explores an existing vulnerability.

II. Related Work

A. Antivirus Diversity

Antivirus solutions are normally diverse between each other, i.e., they do not detect the same thing at the same time [1]. This happens due to a series of different reasons. The first and most obvious results from the fact that they are developed by different vendors with different software engineers and security experts. The second major reason refers to software updates, which are done at different times, with different capabilities. The third reason is due to the solution configuration, as one solution can have a more restrict and tight configuration while other can be more "loose" (i.e., default security policies are different amongst available solutions). Gashi et al. [1] reveal another reason: the regression in the detection capability of the engines. In certain cases, malware detected at a particular version of the engine is no longer identified by newer versions of the same engine, i.e., it is removed from the malware list of that particular engine. This difference can have several reasons but one can speculate that it is either resultant from bad software programming or bad security analysis of the suspicious content. Gashi et al. present also several important conclusions, with two of them having a special relevance to this work. The first is that using more than one antivirus engine, results in an improved detection ratio of malicious software. The other is that from a list of 32 different antivirus solutions, one can combine any two of them and achieve a very high detection ratio. Moreover, 18% of the combinations result in an immaculate detection capability, i.e., every single malware from the tested dataset was effectively detected by one of the two antivirus used. The above conclusions are very important to the work presented in this paper, as it is one of our objectives to present a security solution that can deliver very high rates of detection for malware.

In the solution presented by [2] there are important findings that confirm the need to a different approach concerning antivirus solutions. The first important finding is the fact that the increased complexity of antivirus solutions has been originating a growth in the number of vulnerabilities and is leading to the appearance of malware that exploits these vulnerabilities to infect systems, i.e., bypassing the detection solution by exploiting its own weaknesses. Another important finding confirms the Gashi et al. study [1] discussed above. By applying N-Version programming to malware detection

systems, multiple and heterogeneous antivirus solutions result in an improved capability in the detection of malware.

B. Existing Antivirus Solutions

Antivirus (or anti-malware) solutions continue to be very important security solutions [3]. TrendMicro [4] solutions are a perfect example of the most common deployment schemes for anti-malware in an organization, i.e., it allows to install and configure a set of different engines that will serve different purposes to control and prevent the existence of malware inside the local networks of the organization.

A survey of solutions (specially coming from academic research or some few open source communities) is presented in a recent paper [2]. This work proposes the CloudAV solution, where several different antivirus engines are used in parallel as a network service, improving the detection ratio of the overall solution. This service is presented as a centralized infrastructure to which each host connects to using a multiplatform host agent specially developed to be used in the service. This host agent sends to the network service the suspicious file it needs to check and only after the response from the service will allow or deny access to that file. For the authors, the host agent is not intended to replace host antivirus agents, but instead to work as a complement.

Although this solution has several antivirus running in parallel, there was no intention in creating a fault tolerant system, but instead to just improve the detection capability of the system. That is why there is no clear statement about which algorithm they used to choose the "best" decision, i.e., if there was some kind of a voting phase or how the aggregator machine is picked up amongst all the others in order to execute its task. Other important design feature that is not considered is the apparently single point of failure due to a single aggregator machine that can be vulnerable to attacks, and how attacks can be detected not only in the aggregator but also in other machines that are running the different antivirus solutions.

C. Hybrid Systems

Classical synchrony models in distributed systems and applications are divided amongst synchronous and asynchronous, i.e., whether or not they have time constraints. Hybrid systems appear as a way of dealing with both "worlds" described, allowing a systems architect to increase the reliability of a system. Hybrid systems can be instantiated in several ways [5]. One typical instantiation consists in having a part of the system as an asynchronous sub-system (usually called the payload), which is vulnerable to attacks and executes without any timing assumptions. The other sub-system (usually called the wormhole) only fails by crash and has time bounds, i.e., operates under the synchronous model. In the past there were several proposals of different hybrid systems, from which we mention only three: TTCB [6], CIS [7] and the Delta-4 architecture [8], [9].

Although being an hybrid system, RAVE has a different approach than the referred hybrid systems:

TTCB: Comparing the TTCB approach with the RAVE system we encounter some major differences. TTCB (Trusted Time Computing Base) is not a system, it is a distributed wormhole that provides a series of services: timely timing failure detection, trusted block agreement, trusted random numbers, etc. Differently from the RAVE wormhole, with the TTCB we cannot develop a series of functions and procedures for the TTCB to execute, as it has only a set of minimal functions that it can execute.

CIS: The CIS (CRUTIAL Information Switch) is a kind of distributed firewall that offers a rich access control model and intrusion tolerance capabilities. The CIS operates in a different way than RAVE. It only uses its wormhole to vote the messages that need to be analyzed and the forwarding of the message is done by the payload. RAVE also performs a voting scheme but the wormhole part is responsible for sending the result to the protected infrastructure.

Delta-4: It was one of the first (if not the first) to present the hybrid system concept, with a network interface card (called NAC, Network Attachment Controller) acting as a wormhole for the system which was the computer where this card was sitting on. These cards were connected to the LAN and were fail-silent, i.e., crash-only model. However, and in an opposite direction of the RAVE system, Delta-4 hybrid system does not have its wormholes communicating through a control channel, but instead directly connected to the (payload) LAN. While in the case of RAVE the focus is deploying a replicated antivirus solution as an hybrid system, the focus for Delta-4 is using a special crafted NAC in order to employ an architecture that allows for an host to be attacked, behave in a Byzantine way, but with the wormhole (i.e., the NAC) stopping the LAN to be flooded by packets from uncontrolled hosts.

III. Architecture

A. RAVE System Model

The RAVE system has n replicas, each running a different antivirus. The replicated system is deployed in an internal Local Area Network (LAN) between the Internet and the email infrastructure (i.e., email server or a cluster of email servers), receiving email messages from the Internet that are destined for mailboxes in the domain(s) served by the Email Infrastructure. Replicas are hybrid systems with two parts [5]: payload and wormhole.

a) Payload: It is an asynchronous subsystem composed of $n \geq 2f + k + 1$ replicas in which at the most f can be subjected to arbitrary failures in a given period of time while k can be in recovery as well. Between two consecutive recoveries (concept explained below) if a replica does not fail it is said to be correct, otherwise it is said to be faulty. We assume that replicas' faults are independent, i.e., the failure of one replica is totally independent of another replica failure at the same given time. This assumption is substantiated in practice by using different operating systems and antivirus engines in order to maximize design diversity. Configuration diversity techniques can also be used to substantiate this assumption [10].

b) Wormhole: It is a synchronous subsystem in which at most only f local wormholes can fail by crash. There is one local wormhole per payload replica and we assume that

whenever a local wormhole crashes so does the payload of that replica. The control channel that connects the local wormholes is isolated from other networks, being secure and synchronous.

c) Recovery: RAVE implements two recovery methods: proactive and reactive [11], [12]. Proactive recovery allows the rejuvenation of a system replica, in a periodic manner, with a "fresh" image of both the operating system and application (in the case of RAVE, the application is an antivirus engine). With this we can avoid that an attacker can corrupt sufficient replicas to take control on the decisions of the replicated system. The purpose is to boot up a replica with a different operating system and application than it previously had, removing the attacker advantage of knowing which was the system that previously was running. Reactive recovery is executed by implementing detection mechanisms that can identify compromised replicas and in this way recover them.

B. RAVE Description

Figure 1 presents the architecture of our solution integrated in a global email infrastructure of an ISP or enterprise. The RAVE system sits in between the Internet and the Email Infrastructure, inside the organization Internal Local Area Network (e.g., a DMZ or another less protected internal area). This system has a set of replicas each divided in two parts: payload and wormhole. The payload part is where the antivirus engine is running on top of an operating system and it is the insecure portion of the system. The wormhole is assumed to be secure and receives the application (i.e, the payload) decisions and forwards them to the email infrastructure. In this system the payload parts of each replica are interconnected (including the Internet gateway) by an insecure network which is exposed to WAN communications. Concerning the replica's wormholes, they are connected to the email infrastructure through a trusted LAN. This infrastructure (i.e., the servers) is also trusted and intrusion free, as well are the workstations that are used to administer the infrastructure. The RAVE system has another component which is the Images Repository that contains the operating system and application images that run on the payload at a given time. This repository is used by the wormhole subsystem in order to recover a compromised replica or reinstall a replica based on the proactive/reactive recovery process.

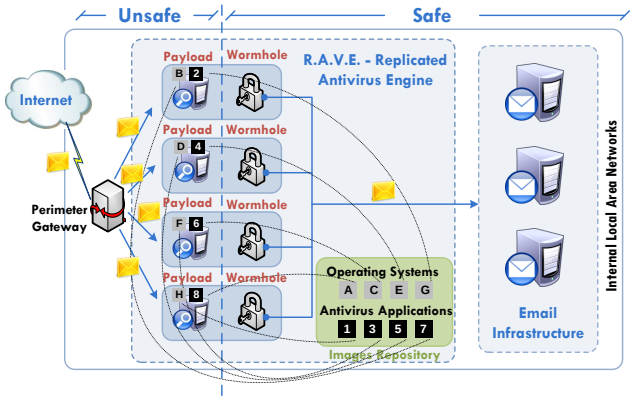

Figure 1. RAVE Architecture.

The perimeter gateway 1 replicates each received email towards all the replicas inside the RAVE system in which different antivirus engines are installed and running in distinct operating systems. Each replica is a hybrid system holding a payload and a wormhole. The payload is where the antivirus engine is running and it is the "entry point" for each email message, therefore it is where messages are inspected for malware. The result of the analysis is then delivered to the local wormhole through a well-defined and secure interface. The local wormholes execute a voting scheme and deliver to the email infrastructure (e.g., email servers) the final output of the RAVE system. The algorithms of both the payload and wormhole are explained in [13]

The **payload** is the part of the system where the email analysis is performed in order to detect malicious contents, i.e., malware. Ideally each replica's payload should have a different operating system and a different antivirus engine at any given time. Looking at Figure 1 we can see that each payload has a different operating system and antivirus engine, represented by a different black letter and a different white number, respectively. Each engine will execute the detection algorithm and return a result (accordingly to a previous configured policy) which will be received by the RAVE payload algorithm and sent to the wormhole through a set of primitives that use a secure and well-defined interface between the two parts. The result, a hash of the result, and the original email are sent to the wormhole. Each engine may behave differently in the case of detection of a virus, or other malware. Some will send an error message back to the originator of the email, others will only remove the threat and send the rest of the email to the internal infrastructure with a small reference to the malicious content found and removed, and some others will just discard the message. Although all these possibilities are true we can configure the antivirus in order to insert some more information that will allow us to normalize the output and send it to the wormhole. This **normalization** is executed because there is the need to convert the result to a format that is understandable by the wormhole. The algorithm behinds this normalization looks at the result of the detection and assign a 1 or a 0, whether a virus was or not discovered.

The **wormhole** has three major functions in this system: one is to receive data from the payload and vote, another is the monitoring of the payload activity and the last one is to execute recoveries based on the detection of problems in the payload (reactive recovery) or based on a recovery schedule (proactive recovery). The wormhole waits for the invocation of one of the primitives available for the payload in order to initialize the procedure to execute a voting scheme. After the voting is concluded, the result is sent by the *master* wormhole to the Email Infrastructure. The role of *master* can be defined simply by looking at the wormhole ID and choosing the lowest one (each wormhole has a predefined ID and knows the IDs of all other wormholes). If this wormhole/replica crashes then the wormhole with the following lowest ID will take the place as *master* (crashes are detected using a perfect failure detector [14]), and so on and so forth. Monitoring the payload activity is important in the detection of faults (benign or malicious). Identification of successive wrong answers (when compared

with all the others) or a pattern of answers (e.g., yes, no, yes, no, yes, no) can be sufficient to detect a fault in the payload. When this detection is made, then it is important to execute a recovery (or rejuvenation) of the suspected replica, more precisely the payload part. In this situation, the wormhole needs to control the execution of the recovery in order to guarantee that the recovery process is executed in a timely fashion and to guarantee that the replica is correctly recovered and starts on a stable and secure state. Proactive recoveries are also controlled by the wormhole subsystem, both in the scheduling of these recoveries as well as in the control of the recovery itself.

The **voting phase** is the core of the RAVE's fault/intrusion tolerance scheme. At this stage all wormholes receive the request, the hash of the request and the output (or response) from the detection engines. When the wormhole subsystem has f+1 different replicas sending the same output, then the *master* wormhole will forward, or not, the email to the Email Infrastructure. With this we can always guarantee that, at least, 1 replica is *correct*, i.e., the output from the detection engine was not changed or in any how conditioned by a fault in the replica itself, as we assume that only f replicas can fail in a given time period. However, it may happen that more than f antivirus engines start to misbehave, e.g., due to a problem during an update. In this scenario, the wormhole subsystem may receive less than $f + 1$ identical responses from the antivirus engines. To address this issue, RAVE has an optional feature that allows the system to make progress even if the wormhole subsystem receives less than $f + 1$ identical responses. When this feature is on, the system administrator needs to specify a timeout (configuration parameter) after which the wormhole subsystem will stop waiting for more responses and make a decision based on the responses already received. The rule used in this decision is also a configuration parameter, and it may state for instance that an email should be approved/forwarded if a majority of antivirus responses says so.

The **images repository** is where the images of the operating systems and the antivirus engines are stored prior to their deployment in the replicas. The major requirements for this part of the system it is that it needs to be secure, intrusion free and available (i.e., communication to the repository is needed at all times, which can be achieved by replicating the infrastructure in several different places). The repository not only holds the images but also guarantees that all of them are updated with the latest stable patches releases, and vendors recommendations. Images in the repository are complete (i.e., operating system with the already installed detection engine) in order to increase the performance of the whole system, more precisely in the recovery process. The wormhole subsystem controls the deployment of images taken from the repository and installed in the replica's payload.

IV. Prototype and Evaluation

The basic idea of the RAVE system was to define a set of building blocks that could be added to the antivirus initial system in order to guarantee fault-tolerant functionalities as well as to improve its basic capabilities, in this case antivirus engines for SMTP traffic. In Figure 2 we can see that the payload module in each replica is divided in two sub-modules, payload IN and payload OUT. This figure depicts the minimum configuration of the RAVE system with $2f + 1$ (i.e., 3 with $f = 1$) replicas each with a payload module and a wormhole. Note that the current version of the prototype does not include recoveries, and so the extra k replicas (Section III.A) are not needed.

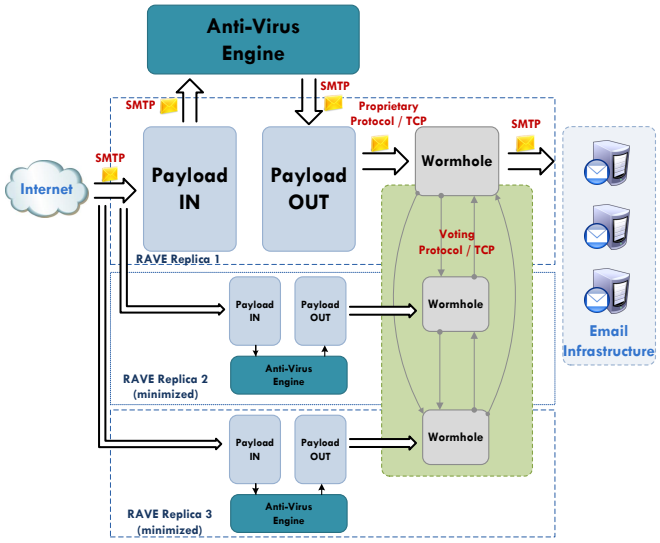

Figure 2. Prototype Diagram (all replicas).

The RAVE system consists of having an antivirus engine running on a machine which is "wrapped around" by a module called payload. The latter is the entry module of the system, as it is responsible of receiving email messages from the Internet, send them to the antivirus engine, receive the message already checked (and possibly changed in case of malicious content being detected) and then deliver it to the second module of the system, the wormhole, which runs on a different machine (the word machine, in this context, refers to an independent computational unit that can be physical or virtual). The wormhole, in turn is responsible of employing the voting scheme between all existing replicas and send the most voted message to the email infrastructure.

Looking in more detail into the system there are two key aspects of it that make it unique compared to other similar replicated systems. The first is the fact that the payload modules do not allow any interaction from and to the antivirus engine from other component without passing through it. This is the application of the "wrapper" concept to our system, which potentially could be applied as an abstract layer to any other solution. Without losing any functionality given by the antivirus engine, we deployed a solution that is in charge of every type of communication to and from the antivirus. The second unique aspect is the "one-wayness" of the system, i.e., email messages coming from the Internet are delivered to the replica payload which, after antivirus processing, are sent to the wormhole that then delivers the most voted message to the email infrastructure.

The two above described aspects are important regarding security as they stop the vast majority of attacks that usually affects this type of systems, especially the ones that are targeted at the antivirus engine and at the communication processes. The objective of the payload is not to stop the attacks themselves, but instead to be a front-end to these attacks, which will not be able to reach the antivirus engine. Clearly the payload IN may crash facing several of these attacks (namely denial of service attacks) but that will prevent the antivirus to crash also, allowing it to continue its processing jobs if any are still in its mailqueue. After the antivirus mailqueue becomes empty the replica will appear to be crashed to its wormhole as it stops sending data to the latter. This situation triggers a reactive recovery.

"One-wayness" of the communication is a dramatic improvement in the security of the whole system, because it simplifies the monitoring and the communication interfaces between modules, especially between payload and wormhole. As seen, the payload is vulnerable to attacks coming from the Internet, while wormholes have an immensely less probability of being intruded compared with the payload. Having a well defined communication interface between these two modules is a first good measure to bound the capability of the possible attackers, although still representing a possible entry point for an attacker to explore. This is why is so important to have the above characteristic, with only one type of message being exchanged from payload to wormhole, using a very specific TCP port and well defined data types. There is no communication from the wormhole to the payload, giving no feedback to attackers in case of malicious attempts.

The prototype was deployed in the LaSIGE/FCUL virtual servers farm and is composed of 3 physical machines all running Xen virtualization software [15]. On each there is a wormhole running Fedora Core 6 executing in Xen dom0 while the payload (also running Fedora Core 6) is executing in a domU guest virtual machine. Because we were using Xen virtualization software there were not a great variety of Operating Systems that we could use, or better, we could use only several different Linux distributions. We used three different antivirus solutions, two are comercial aplications (Trendmicro and Kaspersky) and one is an open source tool (ClamAV).

The configuration that uses the solution from Trendmicro (Interscan Messaging Security Suite - IMSS) [16], running on top of Linux needed an external application as MTA as it does not have integrated in the solution the capability to receive email messages from the Internet and then send them to the email infrastructure. Figure 3 shows the configuration implemented where the MTA used was the Sendmail program, with two instances of sendmail with the corresponding different configuration files.

The Kaspersky solution (Kaspersky Mail Gateway) [17]was installed from a trial license with a 50 mailbox limitation. Figure 4 shows the message flow inside Kaspersky's Mail Gateway. It is a very simple architecture with a single file to configure it, although there are a long series of sections that can be use to tweak the configuration, redefine filtering policies as well as group policies. Its configuration possibilities are very similar to the ones that we encounter in the *sendmail.cf* configuration file, although this latter is far more complicated and, perhaps, complete.

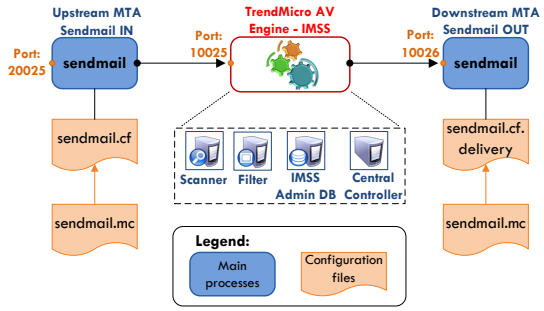

Figure 3. Configuration with Trendmicro using Sendmail.

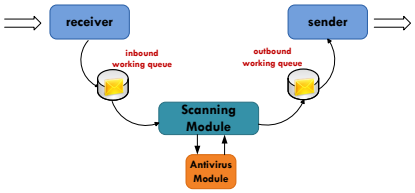

Figure 4. Configuration with Kaspersky.

The configuration using ClamAV solution [18] is depicted in Figure 5 where we can see the four basic processes of this implementation as well as the message flow inside the solution with the sendmail acting as the entry and exit point for each message that needs to be analyzed. It shows the complexity of a system like this which comprises several configuration files for the processes to be up and running. A key feature is the use of Unix sockets (instead of using TCP connections) that allow the communication between the sendmail and the clamav-milter (acronym for mail filter), and from this to clamd that effectively is running.

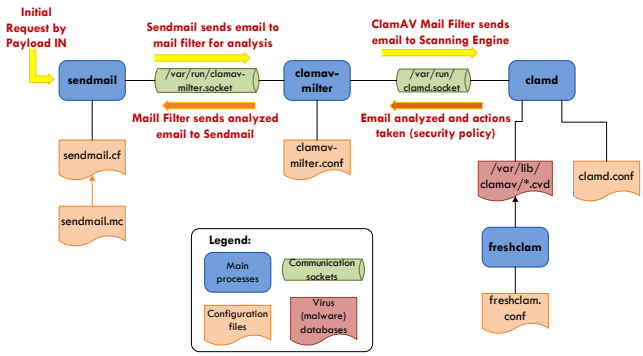

Figure 5. Configuration with ClamAV tightly integrated with Sendmail.

As previously mentioned the prototype was built using 3 different configurations based on 3 different antivirus solutions, which are all extremely different from each other in terms of configurations and default policies. In fact this was a major challenge while building the prototype. Several tests were

executed against the system, in order to evaluate it in different scenarios (only one configuration running, all configurations at the same time and in stress conditions) with and without the voting phase. One of the most important conclusions that we gather from the tests was that RAVE components do not introduce significant latency to the antivirus solutions that are running inside each replica. More precisely, we concluded that the latency imposed by the message scanning and policy checking made by the antivirus engine is responsible for the "lion's share" of the overall latency of the system. Another important conclusion revelead from our evaluation is the fact that when all the engines are working properly then the system presents a better result (in terms of detection efficiency) than any individual engine. This is known because the evaluation was using a set of known emails, with known malware files which were not all known by every engine used. However RAVE was able to detect them all, improving the detection as expected. As well, some of the tests made include a faulty engine which didn't present its result in a timely manner which changed the final result as there is timer to wait for the payload. When that timer expires, and as long as we have a $f+1$ results from the payloads then a decision will be made. For the purpose of this work a default policy was used with no fine tunning for this specific issue as it was not the main objective of the whole evaluation. An extensive and detailed information about the test sets and the results of the evaluation to the system can be found at [13].

V. Conclusions and Future Work

A. Conclusions

It is our opinion that RAVE represents a significative breakthrough for anti-malware solutions. Due to the fact that it uses different antivirus running in parallel and implements a voting phase in order to choose the best and more accurate output concerning the received email (detection capability is clearly improved), it is a system that presents a stronger capability in handling several type of errors like the ones originated by the malfunction of a single antivirus. Also the fact that it can continue to work even under antivirus updates represents an important feature regarding the availability of the system as a whole.

With the development of RAVE we confirmed that fault-tolerant applications can be the next step for several solutions that exist on the market nowadays. It is our belief that development of systems like RAVE can create the necessary momentum for the effective deployment of fault-tolerant solutions in the security solutions arena, without major development costs. This evolutionary path will create the necessary conditions for the whole security paradigm to change from redundant solutions in which the customers need to believe that the solution they are about to purchase is the best fit for their needs, to solutions based on several different services, executing independently and agreeing collectively, enforcing security, accuracy and availability .

B. Future Work

Future work will include the study of other solutions like anti-spam solutions (most of them already present in the antivirus used in RAVE), URL and content filtering, access control and authorization devices. Regarding security solutions there are some that are used to detect bad behaviors or to allow/deny the access to some resource. If systems or applications are willing to wait for decisions during a certain period of time (i.e., without any strong and tight requirement regarding time bounds) in order to execute their actions, then it is our opinion that these systems are amongst the best candidates for a fault tolerant system similar to RAVE, in order to achieve better detection ratios and/or less false positives. A web proxy, for instance, can be a good candidate for a RAVE system if it implements some security policies that define which users can or not access what and when. Having a fault-tolerant web proxy, like in RAVE, would increase availability and security as the result of having more than one machine running the protocol and deciding which user can access to which content and at what time.

References

[1] I. Gashi, V. Stankovic, C. Leita, and O. Thonnard, "An experimental study of diversity with off-the-shelf antivirus engines," *IEEE International Symposium on Network Computing and Applications*, pp. 4–11, 2009.

[2] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," *Proceedings of the 17th USENIX Security Symposium*, 2008.

[3] R. Richardson, "CSI computer crime and security survey," *Computer Security Institute*, 2008.

[4] TrendMicro *http://www.trendmicro.com*, 1988.

[5] P. Verissimo, "Travelling through wormholes: a new look at distributed system models.," *SIGACT News*, vol. 37, pp. 66–81, 2006.

[6] M. Correia, L. C. Lung, N. F. Neves, and P. Veríssimo, "Efficient byzantine-resilient reliable multicast on a hybrid failure model," Oct. 2002.

[7] A. Bessani, P. Sousa, M. Correia, N. F. Neves, and P. Verissimo, "The CRUTIAL way of critical infrastructure protection," *IEEE Security & Privacy*, vol. 6, pp. 44–51, Nov-Dec 2008.

[8] D. Powell, D. Seaton, G.Bonn, P. Verissimo, and F.Waeselynck, "The delta-4 approach to dependability in open distributed computing systems," *18th IEEE International Symposium on Fault-Tolerant Computing (FTCS)*, pp. 246–251, June 1988.

[9] D. Powell, "Distributed fault tolerance: Lessons from delta-4," *IEEE Micro*, vol. 14, no. 1, pp. 36–47, 1994.

[10] A. Bessani, A. Daidone, I. Gashi, R. Obelheiro, P. Sousa, and V. Stankovic, "Enhancing fault / intrusion tolerance through design and configuration diversity," *Proceedings of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS 2009)*, 2009.

[11] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, pp. 452–465, 2010.

[12] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, pp. 398–461, Nov 2002.

[13] C. Silva, "Rave: Replicated antivirus engine," Master's thesis, Carnegie Mellon University and Faculty of Sciences from University of Lisbon, 2009.

[14] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43:2, pp. 225–267, 1996.

[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield, "Xen and the art of virtualization," 2003.

[16] Trendmicro, "Interscan message security suite," *http://emea.trendmicro.com/emea/products/enterprise/interscan-messaging-security-suite/index.html*, 2000.

[17] Kaspersky, "Kaspersky mail gateway," *http://www.kaspersky.com/mailgateway*, 2006.

[18] ClamAV, "Clamav antivirus toolkit," *http://www.clamav.net/*, 1999.