Independent Study Report

Remedium

A cooperative virus defense system

Nuno Brito (nbrito@andrew.cmu.edu)

Carnegie Mellon University | University of Coimbra

Summer Session - August 2010

Introduction

Viruses have been a reality that plagued computer users and organizations since the past few decades. It is a fact that no system is perfect when it comes to protect a machine from malicious actions.

In the past anti virus solutions would rely on databases of known attacks to identify and eliminate attackers and prevent security threats. With the growth of virus variants, the difficulty of indexing them also increased and these products incorporated behavioral and signature based solutions to gain flexibility and higher performance when scanning a given file.

Nowadays, the defense trend is moving to cloud computing environments where an anti virus will learn new threats based on the feedback from the clients installed on desktop machines [12]. But even with this new method, none of these techniques has still been proven effective enough to eradicate threats that menace computer users.

As important as presenting a good report, presenting the reasons that motivate the need for this development allows understanding where and how our project will make a difference to make the world a better place to live.

There is a visible war between anti virus companies and organizations against malware authors. The victims are computer users whose computers are used as weapons by either sides of this conflict.

As in any guerilla war, regardless the size of a given company or organization – it is not possible to effectively fight against an enemy that is anonym and constantly changing its attack techniques.

Cloud computing is a buzz word that has grown popular amongst anti virus products but we propose moving further and adopt a concept of human computing such as the one suggested by Luis von Ahn in CMU during his “Human computing”, term he coined for methods that combine human brainpower with computers to solve problems that neither could solve alone.

We recognize that a system should allow human computing cycles to improve the available methods for handling a threat and secure a system.

This way, we use fire to fight fire and ensure that new threats can be balanced with a solution provided not by anti virus makers but rather by users/machines in contact with the threat. We pick lessons from the past and inspire our structure on a similar way that the Romans were organized, reason why a significant number of Latin terms such as cliens, civis, triumvirate and so forth are present in the project.

We call this project “remedium”: a remedy for computers.

Context

In this section we will explore the environments where this system is intended to become available.

Malware is a definition applied to software intended to infiltrate inside a host computer without the owner's informed consent [1]. After malware settles on the host machine, it may be used to extend the range of malicious actions that serve the perpetrators interests.

Malware can be very specific about the type of target to attack such as noted on the recent Windows link files vulnerability [2] that was focused on industrial control systems, used as an efficient information warfare weapon for industrial sabotage.

On the other hand, the same vulnerability is present to all other machines using Windows as a platform since version 2000 up to the present version 7 and other software authors already began crafting other application exploring the same vulnerability for their own malicious purposes [3].

I've personally experienced in first hand the successful infiltration of the Conficker virus inside a military network [4] in 2008 as head manager of the LAN at one of the operational units. At the time, there was scarce information about this particular threat when it first appeared, thus allowing the virus to spread unopposed to thousands of computers across the country in our military WAN.

This infection caused the LAN under my responsibility to remain offline for a week, time necessary to restore all workstations back to a safe condition and harden our security measures.

In case this had been a deliberate attack to our organization; it would have successfully crippled our defenses as our information flow depends on the network and respective workstations.

Antivirus products exist as early as 1987 [5] and over the past two decades have focused mainly in detecting new variants of viruses with resort on two techniques:

- Signature based detection
- Heuristics

Signature based detection requires frequent updates of the virus signature dictionary from a pool of viruses that continues to increase at each day while heuristics focus on detecting malicious activity to identify unknown viruses.

Keeping in mind recent incidents such as Conficker, Ghostnet, Agent btz or Silent banker, it is relevant that we explore ways of improving network defense to resist new threats.

The possible vectors of attack and defense on a network are quite extent, on this research we are focusing on the same vectors used by Conficker or Agent btz (i.e. dll malware) that corrupts files of the Operative System to perform or allow malicious actions to take place.

On public domain, it is possible to find interesting concepts that address the issue of timely updating virus signatures or heuristics detection based on information gathered from other antivirus clients. I've selected two of the most relevant to our case:

- Cloud antivirus - researchers from the Michigan University proposed in 2008 the concept of cloud antivirus [6] in which files arriving on a given workstation are dispatched to a central location to be analyzed.
- HIDS (Host-based intrusion detection system) [7] is concept where the host monitors and analyzes its own internals. An example of project employing this technique is OSSEC (Open Source SECurity) [8].

There are advantages of using these concepts when compared to the traditional antivirus products.

- A cloud antivirus allows uploading a given file to a server machine with enough hardware conditions to process the file through a batch of antivirus scanners from several vendors.
- This concept eliminates the possibility for a compromised host to sabotage antivirus products [9] that are located on the remote server while reducing the stress imposed on client workstations, as they only need to run a lightweight application to interface with the server.
- After learning the way in which a given operative system typically works, HIDS is capable detecting behavioral changes and trigger an alarm to call the attention of the system administrator.

However, the value of both techniques falls short in certain key aspects.

- A cloud antivirus uses a single server to process the incoming files. This represents a single point of failure that threatens the whole system in case of malfunction or overload of server resources.
- A HIDS is a sensible approach to detect anomalies but it falls short in understand the internals of the operative system. There are vulnerabilities such as the ones explored by Conficker [13] that would pass unnoticed. A fully automated method is always susceptible to evasion techniques from

malware authors. In addition to these false negatives, HIDS tends to suffer from excessive probability of false positives.

For our project we would like to introduce a human component that allows users and network administrators to also take part in our system as a key element on the decision process that exposes anomalies as malicious activities.

We found no project available on the public domain that would allow end users without technical training to actively cooperate on the task of recognizing threats at their host systems. We also wanted to provide a decentralized concept that would be robust enough to withstand malicious attacks.

Our strategy to expose malware consists in creating a knowledge database that focus not on malicious files but rather on legitimate files. New files or files that mismatch definitions found on the database are marked as suspicious and the end user is warned about their existence.

We then collect the data from all clients and determine the reliability of files that are processed. This system is capable of working alongside with existent antivirus on the workstation.

As with any system, there is a risk for true/false positives and negatives to appear, on top of our automated decision components, we allow the users to also add their opinion regarding the reliability of any given file.

In the case of a Conficker or Agent btz attacks on a network equipped with remedium, it would be theoretically possible to expose the tampering of files on the operative system and warn other clients about this attack. Reactions measures would be defined by the systems administrator and could range the immediate disconnection of compromised hosts from the network up to the replacement of compromised files with files from another host considered of trust.

Scope of the project

While presenting the independent study proposal, we divided the project into several phases that would be oriented according to the ACDM (Architecture Centric Development Method) [11] process.

On figure 1 we detail the initial timeline estimation for each week of the semester.

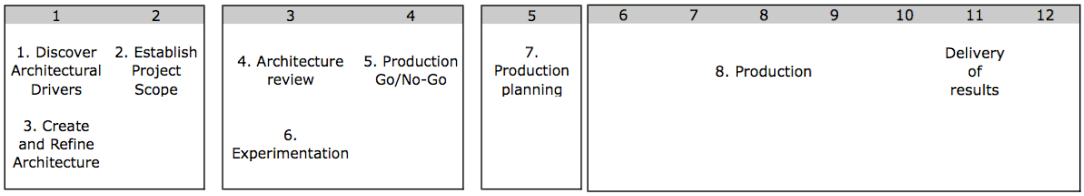


Fig 1 – Estimated timeline

On the first two weeks, time was used to clarify the architecture and introduce new ideas. *(documented on remediumStructure.doc)* The initial stages allowed grasping a more realistic notion of the project dimension. The independent study proposal had previously set specific goals to achieve [14] and time was used to study the technical feasibility and possible routes to be adopted on our architecture.

It turned out that such system albeit simple in nature, is composed with a multitude of roles composed by humans and machines that function as active parts of this system.

With a more realistic notion about the dimension of the project and the inherent time/human limitations, it became necessary to distinguish two relevant scopes:

- Scope of the project;
- Scope of the initial prototype.

The scope of the project comprises the system as a whole detailing the components, details of interaction and tactics for the structure design.

The scope of the initial prototype is a proof of concept key components on our system that were considered as possible to achieve during the available 6 weeks of coding effort.

Scope of the initial prototype

To properly define the scope of our prototype, it was necessary to prioritize the components by order of importance.

Having priorities allowed deciding which components should be implemented since the available resources wouldn't allow completing all the components of the system within the given time frame.

The rational used for priorities on each component can be read on the Remedium Structure document, we prioritized the top three roles to be implemented during phase 8:

- Cliens
- Civis
- Triumvirate

These components are briefly covered over the next sub-section.

Components of the initial prototype

Our project aims to become a modular IDS (Intrusion Detection System) for Networks (NIDS) or Hosts (HIDS) that networks with other participants to strengthen the individual host knowledge using the collective knowledge gathered from other participants.

CLIENS

Cliens is the direct Latin translation for "Client". In the case of our system, each participant is a client where roles are added on top.

Regardless of the prioritization of roles to be implemented, the cliens framework is a dependency that must be made available prior to the development other components so that we can integrate them on top of cliens.

The technical functioning of the component is detailed on the Remedium Structure document.

CIVIS

Given the nature of this project, focus on the HIDS component became the top priority.

Civis is the roman translation of "citizen" and represents our HIDS component.

In the case of our analogy to a roman village, civis is the role of common citizens on the fortified village. In everyday activities, citizens are expected interact with each other inside the village as part of their daily routine activities.

In our case, we assume that no perfect defenses exist and that any civis can eventually turn into a compromised host working for an external threat within our defense borders.

The Civis role as the HIDS component is one of the most challenging components, employing techniques different from the ones currently available on other HIDS projects such as OSSEC, SamHain and Osiris.

Our approach aims to provide civis hosts with a set of automated tools that expose anomalies within themselves, using the human operator of the workstation to decide the type of action based on the exposed evidences when it is in doubt about the action to perform. The novelty is that the anomalies are not in the behavior of the OS, but the idea is to detect the fact that some files have been corrupted, in ways that are impossible to conceal.

A checksum for each recorded file is generated using the SHA-256 algorithm. These records of data are initially stored on a database hosted locally at the workstation to detect if changes occur (functioning as a HIDS) and then transfer upstream the local knowledge to another level of the hierarchy where it is accepted or discarded as useful information that should be made available to other participants.

The remedium system resorts to a collective knowledge repository that is shared amongst all participants of the network. It is capable of distinguishing different versions of the same file under Windows and other desktop OS such as Mac and Ubuntu.

We call this next level in the hierarchy as Triumvirate, which is briefly described further along on this chapter and detailed on the Remedium Structure document.

Looking back to Civis, monitoring different versions of the same file is a key characteristic to prevent the appearance of false positives.

In the event of indexing two files with the same name we would see one of them marked automatically as suspicious. This type of event is common to appear. In the case of the Windows OS it is possible to notice how several versions of the same file are kept in different folders to ensure legacy support. "DLL Hell" [10] is an example of these recurring duplicate events that been acknowledged to exist since the past two decades.

For a typical HIDS, this type of functionality becomes irrelevant as the knowledge is only built using the files located on the host machine, but to meet our goals we aim building a collective knowledge about files that should be considered trusted.

Therefore, ignoring versions of executable files prevents our knowledge from accuracy on the above-mentioned case of two files with the same name but different versions since their SHA-256 checksum would cause a false positive alarm.

In regard to other HIDS already available on public domain, we also noticed that these projects lack a user interface and learning curve adequate for typical workstation users.

Given our goal of proposing a system where the average user decision assumes a significant weight in determining if a detected anomaly is malicious or not, one of the challenges for our prototype is proposing a user interface that can be used without forcing a steep learning curve.

There were several questions about the feasibility of this system that needed to be clarified:

- Can file versions be accurately distinguished?
- Will the software application be capable of running on multiple platforms?
- In terms of RAM, CPU and disk usage – can a typical workstation use our system? ("Typical" is defined on the structure document)
- Can the database support the scaling of usage to millions of data records?
- In terms of performance, how is user-responsiveness of the system affected by the number of indexed data records?
- Is the user interface intuitive enough to be used by non-technology savvy workstation users for them to make informed decisions?

Due to the reasons detailed above, the Civis role becomes the primary goal to implement on the prototype so that we demonstrate in practice that these technological challenges could be met.

TRIUMVIRATE

A triumvirate is a component that serves interface of the cliens with the remaining participants in the network. We can state that a triumvirate assumes the role of Server on a Client-Server configuration where cliens is the client.

However, triumvirate comes from the Latin translation of "three people", more accurately describing three influent people in the society. On the case of our system, each client will interact with a single server but the server itself is contained inside a redundant system of three server machines.

Each single server inside this triumvirate is denominated as triumvir (single person from the group of three people).

The purpose of using a redundancy of three machines is to prevent a single point of failure. A client will connect to a specific triumvir and in case the triumvir fails or is deemed as compromised, another machine is readily assigned to take its place.

To better understand why three is used instead of any other number, the technical implementation of this component is covered in detail at the remedium structure document.

As the last stage of proposed implementations for our prototype, we planned to incorporate part of the triumvirate functionality to provide a better demonstration of the prototype functioning.

This component was deemed as optional on the priority list for the initial prototype in case there was enough time to proceed with development after completing the previous two components.

Lessons Learned

On this chapter we detail some of the lessons that were learned over the course of the independent study and reflect on some of the decisions that were made early on the planning stage and how they impacted on the development effort during the production phase.

What went right

ACDM (Architecture Centric Development Method)

The planning and understanding of the project scope that occurred from phases 1 to 5 of the ACDM were crucial to ask questions about the feasibility for this project while exploring and documenting the architecture.

The experiments made during stage 6 became a valuable asset; it was possible to invest time in exploring the technical feasibility of a given approach for a system feature and then reuse the implemented code on the prototype.

This was the case for the USB detection functionality; it was possible to properly implement and test the feature and then applying it directly on the production unit, saving time on the production phase of the project.

Mentor support

The help from the mentor and other experts in determined fields of technology, throughout each stage of development, professor Benoit Morel kept asking pertinent questions regarding the design decisions and providing feedback regarding the intended behavior of the application in terms of security and anomaly detection.

Other experts were also queried about specific technical details. For example, Professor Marco Vieira is an expert on performance and recovery in database systems and he also helped to solve the technical limitation of performance vs scalability that was found on our database system.

Choice of Java as programming language

My past experience in software development had been exclusively focused on the Windows operative system. I have expertise on internals of this operative system to achieve the intended results but at the same time felt constrained to this platform and this was an opportunity to use a multi-platform language.

Albeit the time invested in ramping up the learning curve to learn this language, it was an interesting challenge and effort that allowed me to grow as a software developer.

Roman theme

Using the roman theme as inspiration for the architecture of our system was also a valuable help. Looking upon the history of human society, it is possible to trace a very realistic level of analogy between the real world and the virtual world.

In human society, malicious actions have always been a constant. Machines, just like humans, cannot be considered perfect and flawless. Understanding this reality helps to keep in mind that each component in our system can be compromised regardless of our security measures.

Using a command hierarchy similar to the one found in roman structures helped to create several layers that minimize the impact of malicious attackers that wish to tamper the system as a whole.

What went less well

Available resources

Coding effort was limited to a single developer. This meant that one single person was responsible for the metrics, working on the implementation of features, perform the quality assessment tests and solve defects. Albeit this was a known fact since the inception of the project, it required more effort to track, plan and monitor the project evolution simultaneously.

Perhaps it would have been more appropriate to work in a team composed with at least two or three elements to ensure that the workload is not so intensive. In average, each week surpassed the expected 12 hours per week devoted to the implementation of this project.

Technology unknowns

Right from the start we tried to identify the areas where it was uncertain the technical feasibility. Experiments were conducted and the prototype itself was also used

It was not possible to predict all technicalities involved with the project. While developing the prototype, there were issues that required further reflection on the approach that should be followed.

- Given the goal of implementing a software application that would work across the mainstream desktop operative systems such as Windows, MacOS and Ubuntu, it was only discovered during implementation that most MacOS machines would only ship with Java 5 as default and this forced to spend more time adding features available on Java 6 but not on Java 5.

Initially, time had been allocated for the development of each selected component on the prototype across the 8 weeks but our custom HIDS, Civis, ended up requiring more time than initially expected. This delay occurred due to several factors:

- Extracting file version from .exe and .dll files was more difficult than initially expected as the version is not listed inside the PE header but rather inside the file as a resource, consuming far more time to implement – it should have been better addressed during the experimentation stage.

- Hot tracking of folders was also not implemented on Java 5 as we were counting on using Java 6 where it was available.

- Time spent on defect fixing passed the initially expected 30% of time available for code development.

- Memory leaks and dimension of files processed. Initial tests would focus on indexing 2000~5000 files but once we started indexing a workstation in field conditions, it would halt after processing 50 000 files. This defect was due to memory leaks that were not anticipated.

- Issues on the database system that was employed. Unlike typical databases that are well adapted to concurrent connections, a flat-file database requires careful handling since each process writing data will lock the database file and other process attempting to write data during this time interval will fail.

Studio project

Another factor relevant to mention is the concurrency between this project and the production of the Studio project. The independent study had initially 12 hours per

week allocated for development while the Studio project had 48 hours of development.

At some point during development, the independent study required more work hours and this was a scarce resource as most of the time was already used by the Studio project.

Involved complexity

Last but not least, after the initial 4 phases of the ACDM, we acquired a more realistic notion of the effort involved. The complexity of the project would require more time and resources than those available until the end of the semester.

Although a satisfactory rate of progress was accomplished and the project itself is innovative, there is a still feeling of lesser accomplishment by not delivering a fully working prototype at the time of this report.

What could be improved

Expert judgment

It was not an easy task to address some of the features required on this project. The components were prioritized but the features inside each component should also been evaluated in terms of perceived difficulty to achieve using expert judgment.

Though not being a perfect method, it would have allowed to better decide if an experiment was necessary before moving to production and evaluate other alternatives if possible, or at the very least grasp some advance in terms of knowledge about the coding specificities of the feature. This would have likely saved time during the production stage.

Metrics

Better metrics tracking, during the initial weeks it was noted that the tracking of defects and time spent correcting them was not controlled with rigor. On future projects I would begin by defining from the start how defects should be tracked, analyzed and prioritized.

Experiments

We tried to define the most relevant experiments before entering production. However, the level of unknown technologies with which we are dealing was quite extensive and these factors delayed the estimated production rate:

- Using Java as programming language:

    - Initial learning curve;
    - Inexperience creating and handle a UI in Java.
- Implementing the code to:
    - Read PE headers;
    - Tracks changes on monitored folders / USB drives.

In the future, it would be wise to assess the level of unknown technology and define a more extensive number of experiments to reduce the risk of delay or missed estimations.

Metrics

This section focus on the metrics collected during the development of the project.

Development effort was categorized into four distinct types:

- Meetings (MEET) – All meetings with the mentor and experts interviewed.
- Documentation (DOC) – Tasks related to the documentation of the architecture, the elaboration of the present report and performed reviews.
- Coding tasks (CODE) – Time focused exclusively on the code implementation of components and experiments of this project.
- Defects (DEFECT) – Time used in detecting and correcting defects found on the project.

In terms of defects, there is no distinction between the types of defects, Due to the limited resources; we decided to make no distinction between the severity levels.

Initial planning

On week 7 at phase 5 of the ACDM there was the opportunity to define a percentage threshold for each type of activity, however, given the level of technology unknowns in our system it was decided to reevaluate progress at each week to ensure an adjusted response to the project development.

Hours in documentation and meetings

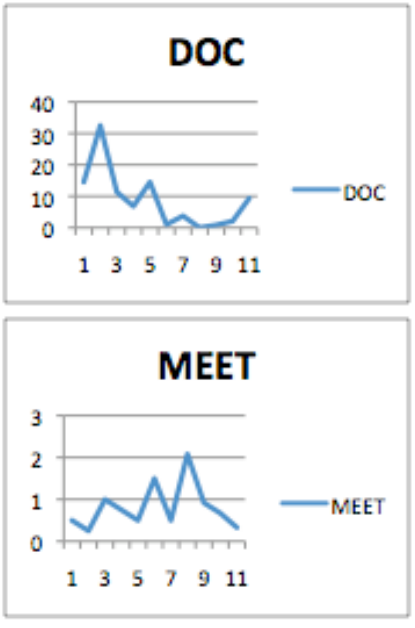
The initial phases of development were focused on the design and documentation of the architecture, reason why they are higher on the initial three weeks.

The time available for research consisted on twelve weekly hours but it became necessary to invest more time into a deeper investigation of the project scope.

Over the following weeks, documentation and the report were reviewed; reason why the graph presents a rise in hours used for documentation after week six.

In terms of meetings, they were more frequent during the final reviews of the architecture and during the expert interviews. After weeks eight and nine, the feedback rate increased as more components became available for review and reevaluation.

Fig 2 – Hours spent in documents and meeting at each week of development



Hours in coding and defects

Coding began at week 3 with experiments to explore some of our tactics for the architecture. As prescribed by the ACDM, we moved to phase 8 of production after week 6.

Across the following weeks 5 to 9 it is possible to note the high focus given to coding activities.

In terms of defects, they were occasionally detected and corrected as soon as possible. Given our time and human constraints, it was not possible to conduct a more intensive quality assessment of the produced code as desired initially.

On week 9, it was decided to focus on the code quality. A significant amount of time (around 25 hours) was allocated to review and verify if the code met the expectations for each component.

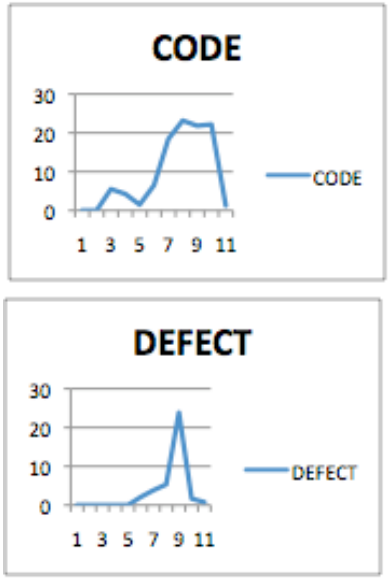


Figure 3 – Hours of coding and defect handling per week

Performing an intensive analysis of the project at this week was crucial to improve code quality in terms of readability, structure of components and simplification of methods.

This tactic proved to be efficient, focusing two weeks mainly on code development and the following week on code quality ensured that we balance both tasks with enough quality time.

Otherwise, there would be a risk that we neither perform an intensive quality assessment nor have enough time to explore possible routes of implementation for new components when the technology presents a significant level of uncertainty.

Reflection about time distribution

Table 1 presents a more detailed view of the hours used for each type of activity across the weeks.

The last line of this table indicates the impact in terms of percentage that each activity had on the overall development effort.

81% of the used time was invested in documentation and coding tasks.

Week	MEET	DOC	CODE	DEFECT	SUM
1	0,5	14,54	0	0	15,04
2	0,25	32,5	0	0	32,75
3	1	11,33	5,5	0	17,83
4	0,75	6,83	4,25	0	11,83
5	0,5	14,5	1,5	0	16,5
6	1,5	1	6,5	2,08	11,08
7	0,5	3,75	18,42	3,83	26,5
8	2,08	0	23,17	5,33	30,58
9	0,92	0,83	21,83	23,83	47,41
10	0,67	2,08	22,17	1,67	26,59
11	0,33	9,33	1,25	0,75	11,66
SUM	9	96,69	104,6	37,49	247,77
%	4%	39%	42%	15%	100%

Table 1 – Hours spent per week on each type of activity

Analyzing the table it is possible to note that a total of 247 hours were used for development while the available time for this project were 132 hours.

This difference represents an effective average use of 22 weekly hours instead of the estimated 12 weekly hours.

In case a client would be paying to see a product, his expenses would have doubled after the initial estimation and this fact was taken into consideration since the early inception of the project.

This fact was noted after acquiring a more realistic notion of the project scope and effort involved to make it possible. Under an industry scenario, the time line for delivery of this project with the current human resources would have been renegotiated or not pursued at all.

Therefore, I've decided to follow the spirit of an independent study as an opportunity to improve my own professional skills and work as much as possible even at the risk of surpassing the available 12 hours.

Given the fact that we were implementing a prototype, a substantial amount of effort was given on the documentation of the system architecture so that it can be conducted and completed on future developments.

From a total of 142 hours used for coding tasks, 38 hours were used for correcting defects found on the components. This marks an average of 26% in the global coding effort.

Lines of code

On table 2 we provide the final results in terms of Total Lines of Code (TLOC), the number of Java Packages, Classes, Methods and defects that were tracked.

TLOC	3588
Packages	4
Classes	37
Methods	124
Defects	23

Table 2 – Project code counting

The TLOC measure doesn't reflect the lines of code that were modified or removed during implementation.

In terms of effective coding productivity, 142 hours were dedicated to code and defect handling throughout the semester, meaning that an average of 25 Lines of Code were produced for each hour of implementation.

In terms of defects, figure 4 provides a relation of the number of defects found at each week. We start the counting at week 6 as it pertains to the period when stage 8 of the ACDM (Production) debuted.

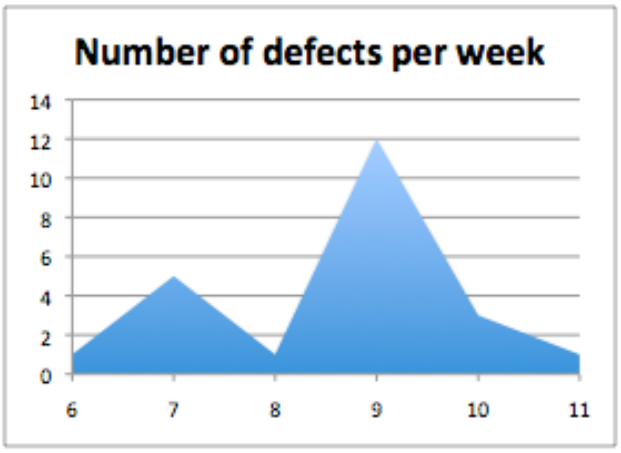


Figure 4 – Number of defects per week

Not all defects are mentioned on figure 4, only defects that were not detected during implementation or could not be solved within the same day of their detection were reported and accounted as effective defects.

This measure allowed reducing the overhead of reporting all defects of minor severity and still keeps track of time spent on the correction of more severe issues.

As mentioned previously, week 9 was used almost exclusively for tasks of code quality. During the code reviews, a significant number of defects were detected and corrected. Across the following weeks the number of defects gradually reduced due to these quality improvements on the core structure.

Development Environment

On this section we detail the tools that used to implement this system.

As coding environment, Java was selected. Since it was intended to run across Windows OS, MacOS and Ubuntu Linux this was a well-adjusted decision.

To code in Java language, the latest version of the NetBeans IDE (6.8) was chosen due to it's default support for the development of UI based applications.

There was a limitation of using only Java 1.5 as it is the only version available as default under MacOS. Inside NetBeans, the profiler tool was used to analyze and correct memory leaks inside the application.

To track time, a Google docs spreadsheet was used. On this spreadsheet, the time and type of tasks were detailed at each day through manual input of values.

In order to exchange feedback with Professor Benoit, Skype was used to perform videoconference calls besides the frequent email messages to ask questions, send documents and receive feedback.

Testing of the application was performed on MacOS and Windows OS machines workstations.

Steps for the future

Following the initial architecture design, this project and research cannot be considered as complete.

The next step is finishing the implementation of the components described on the remedium architecture document. Alongside with the implementation, it is important to periodically review the design and reevaluate if changes should be made.

We have prepared the architecture from a theory level and during implementation, it was noted that many details differed substantially from a level of feasibility and required resources. This was the case for the interpretation of file version on the PE header of executables files on the Windows OS.

During the next semester, the project development will continue and culminate on the delivery of a second prototype that should include the implementation of networking components and field-testing under realistic conditions of operation.

This development will not occur under the subject of an independent study research but the mentor, Benoit Morel, has already demonstrated availability to aid in the guidance and progress of this project in the future.

Conclusion

As a final reflection, I consider that this research was important both at the professional and personal level. The results from this work can also be considered as innovative regarding the way in which anomalies are currently detected.

On a professional level, it was an opportunity to adopt a new language that is relevant on the software industry, to code an application that works across different platforms and propose a concept of defense that is tightly coupled to human intervention.

While doing so, it was also interesting to apply ACDM as a software development method. Albeit not perfect, it provided discipline and method to work within a tight budget of time and human resources.

On a personal level, during the course of this research, there was an opportunity to not only propose a new system but to also learn more about the already existent systems and current threats.

A project of this kind has the potential to scale onto a large dimension of human participation that surpasses the efforts of any single antivirus vendor. This is our aim for the next times: To create a system maintained not only be the original developers but also open for users to engage into an active role in virus detection.

We live in an age where the increasing dependency of workstations to networks makes everyone vulnerable to an increasing number of attackers once a new flaw is explored. Thus, this fact raises a critical need of developing cooperative tools that are capable of responding to (yet) unknown threats.

Our best hope is that this system can indeed become useful to many others, as we will certainly work to make it a memorable milestone in terms of virus defense.

References

[1] Atkinson, J. (2006) Private and Public Protection: Civil Mental Health Legislation, Edinburgh, Dunedin Academic Press

[2] Peter Bright. New Windows Shortcut zero-day exploit confirmed. http://arstechnica.com/microsoft/news/2010/07/new-windows-shortcut-zero-day-exploit-confirmed.ars

[3] Brian Prince. Windows Vulnerability Targeted by More Malware. http://www.eweek.com/c/a/Security/Windows-Vulnerability-Targeted-by-More-Malware-442601

[4] John Moore. Conficker threats military networks. http://www.articlesbase.com/security-articles/european-military-facilities-wage-war-with-threatening-computer-worm-1270371.html

[5] Joe Wells. Timeline of antivirus research. http://www.research.ibm.com/antivirus/timeline.htm

[6] CloudAv homepage. http://www.eecs.umich.edu/fjgroup/cloudav

[7] Wikipedia. Definition of HIDS. http://en.wikipedia.org/wiki/Host-based_intrusion_detection_system

[8] OSSEC homepage. http://www.ossec.net

[9] SOPHOS. Virus example that disables local antivirus defenses. http://www.sophos.com/security/analyses/viruses-and-spyware/w32gonera.html#table4

[10] Wikipedia. Definition of DLL hell. http://en.wikipedia.org/wiki/DLL_hell

[11] Lattanze, Anthony J. Architecting Software Intensive Systems A Practitioners Guide. Boca Raton: AUERBACH, 2008.

[12] Jon Oberheide, Evan Cooke, Farnam Jahanian CloudAV - N-Version Antivirus in the Network Cloud. Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109

[13] Microsoft. Microsoft Security Bulletin MS08-067 – Critical Vulnerability in Server Service Could Allow Remote Code Execution (958644). http://www.microsoft.com/technet/security/Bulletin/MS08-067.mspx

[14] Nuno Brito. Independent study proposal: Cooperative Virus Defense System. Cooperative_virus_defense_system.pdf