Design of an Automated Malware Analysis System

Bradley J. Nabholz

Purdue University - Main Campus, bnabholz@gmail.com

Follow this and additional works at: http://docs.lib.purdue.edu/techdirproj


This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
Design of an Automated Malware Analysis System

In partial fulfillment of the requirements for the Degree of Master of Science in Technology

A Directed Project Proposal

By

Bradley J. Nabholz

March 30th, 2010

Committee Member                                      Approval Signature                                      Date

James E. Goldman, Chair                               ____________________________                   ______

Jeffrey L. Brewer                                      ____________________________                   ______

John A. Springer                                       ____________________________                   ______
# Table of Contents

Introduction..................................................................................................................................... 3  
Statement of the Problem ................................................................................................................ 5  
Significance of the Problem ............................................................................................................ 6  
Statement of the Purpose ................................................................................................................ 7  
Delimitations................................................................................................................................... 7  
Review of Literature ....................................................................................................................... 7  
  Prior Research ............................................................................................................................. 7  
  Effects of Automation ............................................................................................................... 14  
  Private Analysis ......................................................................................................................... 17  
Methodology ................................................................................................................................... 19  
Time Action Plan .......................................................................................................................... 20  
Systems Context Diagram ............................................................................................................. 21  
Detection and Analysis Process Flow Diagram ............................................................................ 22  
Use Cases ...................................................................................................................................... 23  
System Objectives ......................................................................................................................... 28  
  Primary Objectives .................................................................................................................. 28  
  Other Objectives ..................................................................................................................... 30  
  Limitations ............................................................................................................................... 30  
System Architecture Diagram ....................................................................................................... 31  
  Workflow ................................................................................................................................. 31  
Functional Requirements .............................................................................................................. 32  
Development Strategy ................................................................................................................... 33  
Reference Implementation ............................................................................................................. 35  
Recommendations for Future Research ........................................................................................ 38  
Conclusions ................................................................................................................................... 38  
References ..................................................................................................................................... 39
Introduction

Malware is a relatively new term – but examples of malware have existed for decades. Simply put, malware is malicious software written for the purpose of causing computer-related damage. Viruses, worms, trojans, spyware, adware, and targeted malicious code all fall under the umbrella of malware. It’s clear that intentionally inflicting damage, whether in the real world or the digital world, is something that needs to be prevented. However, the only certain way to eliminate the threat of malware would be to eradicate it completely, a noble yet certainly impossible task for any single person to undertake. For the moment, malware is an everyday tool for criminals to inflict damage via denial of service attacks, data ransoms, and financial fraud. To understand the problem that malware causes, we need to look at who is writing malware, and why they are writing it.

A wide variety of people have an interest in writing and proliferating malware. The gains they are seeking may be personal, political, or financial. There may be a direct monetary benefit associated with writing and releasing the malware (common in the financial industry), or there may be only negative monetary effects (lost revenue due to computer damages). In either case, the act of one entity inflicting harm on another is grounds for involving law enforcement, regardless of the method used. However, in “real life” scenarios, law enforcement has clearly defined roles and procedures for determining who, what, where, when, and why a crime has occurred. There may be unknowns, but there are even procedures for handling those as well. When in the context of a computer-based crime, the number of possible perpetrators and attack vectors is staggering. Nonetheless, law enforcement does have procedures for dealing with a number of computer-based crimes.

When a computer-based crime has been committed, law enforcement must determine who was involved or affected, as well as which computers were involved or affected. Through the course of law, computers may be seized and analyzed for evidence of the crime. For most investigations of computer-based crimes, this analysis consists of searching for specific keywords or unusual data on those computers. With the recent rise in number and severity of malware attacks, law enforcement is realizing the significance of both detecting the presence of malware, as well as determining which actions or effects the malware has on a computer and the computers to which it is connected. With severe constraints on both equipment and labor, many
law enforcement agencies are looking for help to solve these malware-related cases. Additionally, the case load of many law enforcement organizations hinders many employees from gaining the expertise to perform thorough malware analysis (Professor James Goldman, Purdue University, personal communication, Summer 2009). For the Indianapolis division of the FBI, the Purdue University Malware Research Team offers expertise and manpower to detect and analyze malware used in computer-based crimes.

The Malware Research Team (herein MRT) operates as a plug-in step to the FBI’s process of investigating a computer based crime. After collection of raw evidence from affected computers in the form of bit-for-bit copies or disk images, the investigator will search the computer for keywords or other specific evidence. If malware is suspected to have played a role in the crime, the investigator will obviously have an interest in finding out which malware exists on the computer and what the role of that malware is. To aid with this, the MRT is given a brief description of the case and a copy of the disk image (inputs to the system), performs a malware analysis process (to be described further in the review of literature), and returns a report to the FBI to assist them with solving the case.

The Malware Research Team has an internal process that is used to examine cases in a predictable, sustainable, and repeatable manner. This process consists of two major phases, which correspond to the primary requirements from the FBI: detection (what malware exists?), and behavior analysis (what does that malware do?) A flow chart detailing the MRT’s interaction with law enforcement, processing of the case, and reporting of results is shown below in Figure 1.
Figure 1 - Flowchart of current malware analysis process

**Statement of the Problem**

While the process shown above in Figure 1 works well for analyzing malware, at an implementation level there are a number of issues which hinder the ability of the MRT to process cases accurately and rapidly (James Goldman, personal communication, Summer 2009). These
issues fall primarily into two categories: reliability and scalability. The malware analysis process and system need to be more reliable and more scalable than they are at present. The following are a list of areas for improvement and which primary system aspect they affect.

<table>
<thead>
<tr>
<th>Area for Improvement</th>
<th>Objective Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability of system components to be validated</td>
<td>Reliability</td>
</tr>
<tr>
<td>Labor-intensity of current process</td>
<td>Scalability</td>
</tr>
<tr>
<td>Privacy of case-related data</td>
<td>Reliability</td>
</tr>
<tr>
<td>Integrity of reported results</td>
<td>Reliability</td>
</tr>
<tr>
<td>Frequency and severity of human errors</td>
<td>Reliability</td>
</tr>
</tbody>
</table>

**Significance of the Problem**

As mentioned above, two improvement categories exist: reliability and scalability. Reliability is important to the Malware Research Team and its customers, because providing accurate information to law enforcement will help to ensure that justice is served. However, scalability is a far more important problem, because a more scalable system would allow a greater number of malware-related cases to be processed by the MRT, without the need for additional personnel with the expertise required for such investigation. Additionally, a system that can scale via hardware instead of via added labor will allow case completion time to become more predictable and consistent. At present, the human-oriented nature of the malware analysis process has resulted in approximately twelve cases being completed in a period of twelve months (James Goldman, personal communication, Summer 2009). Designing a more scalable system will allow the MRT to process more cases, in less time, for both their existing customers and new customers in the future.
Statement of the Purpose

To address the future reliability and scalability needs of the Malware Research Team, a new system will be designed to address the shortcomings described in the Statement of the Problem. Upon completion of the project, a detailed design specification including the following will be delivered to the Malware Research Team:

- Systems Context Diagram
- System Activity Diagram
- Use Case Scenarios / Narratives
- System Objectives and Scope
- System Architecture Diagram
- Functional Requirements and Constraints
- Development Strategy

In addition to the design specification document, a document will be provided which details a method for evaluating the future system against the criteria (areas for improvement) described in the Statement of the Problem.

Delimitations

Reverse engineering and code analysis is very labor-intensive, and while it may be added to the MRT’s analysis process at some point, addressing these issues is outside of the scope of both the current system and the future system.

Review of Literature

Prior Research

While published works describing malware and the analysis of malware are very widespread, works describing end-to-end systems for automated analysis of malware are comparatively scarce. Therefore, much of this section will discuss prior research done by the
author in conjunction with the Malware Research Team. This prior research constitutes a large body of knowledge about end-to-end malware analysis. In effect, the details of what this system does (as well as what it does not do, or does not do well) will feed into the requirements gathering process for the proposed system.

A good way to look at the scope of work done by the MRT is to look at the malware analysis process first as a “black box,” and examine the inputs and outputs of the system before analyzing the process in detail. Two things need to be provided by the end-user who is utilizing the MRT:

- Case description – background information, official write-ups, suspected attack vectors, personnel interviews (if available).
- Case materials – image or other bit-for-bit copy of affected disk drives, or any other digital media needing analysis.

In return, the MRT provides the following:

- Case report – summary of findings (what malware, effects of malware) as well as detailed descriptions of each malware found.

These are important points to bear in mind while examining the process that the MRT currently utilizes – every action should be traceable to the final objective of reporting the malware found on one or more targets.

As shown in Figure 1 in the introduction of this document, two distinct sub-processes occur during the overall malware analysis process, detection and analysis. These two processes correspond to three phases of the malware timeline, shown in Figure 2: scanning and identification, isolation and extraction, and behavior analysis. Also shown in the malware?
The first sub-process from Figure 1 to be examined is detection of malware on the target. Fortunately, there is plenty of off-the-shelf software available for malware detection. Companies like Symantec, McAfee, AVG, and many others produce both free and commercial anti-virus, anti-spyware, and anti-malware applications, which receive frequent updates for catching the latest malware “in the wild.” Knowing this, an obvious approach would be to install a piece of anti-malware software, attach the target media, scan it for malware, and record the results.

However, a number of issues exist which must be considered:

- False positives / negatives – what if the selected anti-malware tool falsely detects malware, or misses malware that exists?
- Preservation of target media – what if the selected anti-malware tool automatically removes any found malware (preventing later extraction), or otherwise modifies the contents of the target?
- Scalability – how many concurrent scans can one piece of MRT hardware perform?
- Security – what is preventing any malware on the target from corrupting or infecting the hardware on which the scan is being performed?

These issues limit the speed at which the detection process can be completed. However, the MRT has utilized various tools to address each one of these issues, which are discussed in the table below.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>False positives/negatives</td>
<td>Multiple scanning tools</td>
<td>Scanning with multiple tools limits the probably of a false positive or negative, via correlation of multiple results.</td>
</tr>
<tr>
<td>Media preservation</td>
<td>Virtualization / virtual disk</td>
<td>Instead of a physical disk, a disk image representing a bit-for-bit copy of the original media is used. This image can be copied for easy backup, or marked read-only to prevent modification.</td>
</tr>
<tr>
<td>Scalability</td>
<td>Virtualization</td>
<td>Virtualization allows multiple operating systems to execute within a host operating system.</td>
</tr>
<tr>
<td>Security of host</td>
<td>Virtualization</td>
<td>Virtualization also isolates the virtualized “guest” operating systems from each other as well as from the host, preventing damage to anything outside of the guest.</td>
</tr>
</tbody>
</table>

Even with the solutions to these problems that the current MRT detection process uses, there are still some limitations and shortcomings which need to be examined. These limitations were described earlier in this document as “areas for improvement.” Below, each area and the specific problem will be discussed.

- Ability of system components to be validated – although the false positive/negative issue is resolved by using multiple tools, there is still a possibility of a user not following procedures correctly, which could cause inaccurate results.

- Labor-intensity of current process – the user who is performing the detection process must frequently monitor the state of in-progress scans, and manually handle result collection and error correction throughout the process.

- Privacy of case-related data – scanning reports need to be copied to a user’s computer for aggregation, resulting in a possible breach of privacy if the information is not carefully controlled.
- Integrity of reported results – the reported results have to be correlated by hand, using (at best) a word processor to organize and condense scanning report from multiple software tools.

- Frequency and severity of human errors – related to validation ability, there is presently no way to ensure that a human mistakes do not happen, or that they are detected when they do happen.

### Table 2 - Summary of solutions for detection process issues

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability of system components to be validated</td>
<td>Easily testable components</td>
<td>Taking a systems approach and redesigning the detection system as interoperable parts will allow each segment to be independently verified.</td>
</tr>
<tr>
<td>Labor-intensity of current process</td>
<td>Automation</td>
<td>Automation will reduce the repetitive prepare-scan-collect process into a one-step “click and forget” process.</td>
</tr>
<tr>
<td>Privacy of case-related data</td>
<td>Automation</td>
<td>Automation prevents human contact with relevant data from the scanning process, allowing better control over that information.</td>
</tr>
<tr>
<td>Integrity of reported results</td>
<td>Automation</td>
<td>Additionally, if the data cannot be easily or accidentally accessed by a human, it cannot be easily changed either, preserving integrity</td>
</tr>
<tr>
<td>Frequency and severity of human errors</td>
<td>Automation</td>
<td>Eliminating the human factor from the process will reduce the possibility of human error.</td>
</tr>
</tbody>
</table>
As seen, a crucial feature of a new system will be automation – this solves many of the problems with accuracy, integrity, privacy, and validation ability which exist currently. However, at this point only the first half of the process has been examined – the detection and accurate identification of malware. Once malware has been determined to exist on the target media, a second process is used to analyze the behavior of that malware.

Although the detection of malware uses software that many computer users are well aware of, malware analysis is much more specialized. A number of online submission-based websites exist which will accept uploaded malware, execute this malware in a controlled and monitored environment (often using virtualization), and then return details about how the malware affected its environment while it executed. Examples of these sites are CWsandbox and Norman Malware Analyzer. A simple approach for malware analysis would be to upload malware to an online analysis service, and then read the results afterwards. This approach causes a number of problems, detailed below.

- False positives / negatives – what if the selected malware analysis service falsely detects an executable as malware, or isn’t able to detect an executable as malware?
- Labor-intensity of current process – what if the malware analysis service doesn’t actually make a malware-or-not determination, but only provides information for the user?
- Privacy of case-related data – clearly an online submission service is requiring malware (from active cases) to be sent over the Internet and stored (if only temporarily) on the computer performing the analysis.
- Integrity of reported results – there is no way to be certain that the results from a malware analysis service were not modified in transit, or that the results are mixed up with another concurrent analysis.
- Frequency and severity of human errors – a human could submit the wrong piece of malware to the analysis service, or otherwise confuse the results.
The MRT is aware of these problems, but many of them are side-effects from relying on a third party analysis service. Because of the lack of control over these services as well as a lack of validation ability, only one of these problems can presently be addressed:

Table 3 - Presently-addressed issues with analysis process

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>False positives / negatives</td>
<td>Multiple analysis services</td>
<td>Submitting suspected malware to multiple analysis services limits the probably of a false positive or negative, via correlation of multiple results.</td>
</tr>
</tbody>
</table>

To summarize - the current MRT malware detection process has in some way addressed many of the initial concerns, except for scalability. However, the analysis process has yet to address many of the concerns described above. It will be the goal of the new system to solve these remaining issues.

Table 4 - Remaining issues with analysis process to be solved

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labor-intensity of current process</td>
<td>Automation</td>
<td>Automation will reduce the repetitive prepare-scan-collect process into a one-step “click and forget” process.</td>
</tr>
<tr>
<td>Privacy of case-related data</td>
<td>Use private analysis service</td>
<td>Implement a local version of an online malware analysis service.</td>
</tr>
<tr>
<td>Integrity of reported results</td>
<td>Automation, Use private analysis service</td>
<td>Perform analysis locally, and also prevent human interaction with the analysis process.</td>
</tr>
</tbody>
</table>
| Frequency and severity | Automation                  | Eliminating the human factor from the
Effects of Automation

Not surprisingly, automation is a large part of the solution for making the analysis process work more smoothly. However, a key point is the introduction of a private analysis service. This service would work much like an online service such as CWsandbox, but would run on local hardware in a secure and controlled environment. This provides a number of improvements in the areas of privacy and validation ability, but also makes it substantially easier to tie into for automation purposes. Scalability also becomes easier to achieve.

To understand the importance of automation on the case-work that the Malware Research Team performs, it’s crucial to look at the balance of time between user interaction (a human physically interacting with the system) and system wait time (wall clock hours the system uses to perform “background” tasks). Shown below is a linear graph of the estimated time for a typical case, in both the detection and analysis processes. The figures used for the time study graphs in this proposal were provided by the Malware Research Team in Fall 2009 for a case which is representative of typical malware analysis cases.

![Figure 3 - Time study of typical detection process](image)

The numeric callouts on the time study illustrations denote what the user or the system was doing during that span of time. Red areas indicate time spent by a human to interact with the system and configure it for the detection process. Blue areas indicate time spent waiting for the system to complete a task. A summary of the tasks denoted in the illustration are listed below:
1. (interaction) Reading case information, setting up a work environment, and beginning to copy the disk image into the system for analysis.
2. (wait) Waiting on the system to copy the disk image.
3. (interaction) Configuring the first detection tool and beginning the first detection scan.
4. (wait) Waiting for the detection tool to scan the disk image for malware.
5. (interaction) Collecting results from the previous scan, configuring the next detection tool and beginning the next detection scan.

   (steps 4 and 5 are repeated for each malware detection tool that is used)

6. (interaction) Collecting results from the final scan, correlating those results to eliminate false positives and negatives, and compiling a malware inventory report.

   Overall, the red areas represent nearly 3 hours of user interaction time required to complete a multi-tool detection run on a single malware-related case. Worse yet, these 3 hours do not occur sequentially – instead a user must come back every few hours to check on the status of the previous scan and begin the next scan. In total, the red areas represent 5 instances of user interaction that would ideally be eliminated, by having the user interact with the system only at the beginning and end of the detection process. The blue areas represent over 9 ½ hours of time spent waiting on the system. And again, this time is sequential – detection scans are not executed concurrently.

   Although the overall wall clock time of 12 ½ hours may seem daunting, for most purposes it is irrelevant. The key factors are the number of times the user needs to interact with the system, and the overall time that a user needs to interact with the system. These are the important figures which need to be reduced, ideally reducing number of interaction points to 2 (beginning and end) and reducing overall interaction time to half of its present value (1.5 hours).

   However, the illustration above only shows part of the story. Following the detection process and the creation of a malware inventory, a decision is made: if the detection process yields no detected malware, then the Malware Research Team may opt to investigate the system manually (time-consuming), or otherwise abandon the process and report that no malware was detected on the target system. If malware was found, then the analysis process begins.
The analysis process is in-depth and time-consuming, similar to the detection process, but there is a difference in the relationship between interaction time and wait time. Whereas the detection process features several large chunks of work to be done (waiting for detection tools to complete their scans), the analysis process features relatively quick pieces of work that need to be completed on all of the malware listed in the malware inventory report. From discussions with members of the MRT (Fall 2009), a typical case may have over a dozen pieces of malware requiring analysis, with up to three separate online analysis services. As shown below, the analysis process looks significantly different when examining time usage.

The striking difference between the two figures has to do with units of work and their relative size. Scanning a typical multi-gigabyte disk image with a detection tool takes a significant amount of time. Submitting a piece of malware (generally less than one-hundred kilobytes) to an online analysis service takes on the order of minutes to complete. However, several minutes are also required to prepare the sample, and several more are required to collect the results and store them. Overall, the following tasks and task groups occur during the analysis process:
1. (interaction) User must interact with the system, view the malware inventory report (created in the detection process), and then prepare an environment to extract malware from the target system and submit it for analysis.

2. In this example, twelve pieces of malware were found, so in sequence, each one is extracted from the target system, and then submitted to an online analysis service via the web (interaction). The analysis service processes the malware sample (wait), and notifies the user when complete so that the results can be acquired and stored in an organized manner (interaction). This process is repeated for each analysis service being used.

3. (interaction) After all of the malware samples have been analyzed by all of the desired analysis services, the individual malware reports from each service must be correlated. This forms the basis of the final report which is returned to the requestor of MRT’s services.

By comparison to the detection process, the analysis process is significantly less demanding in terms of elapsed, wall clock time. However, the demand on the user to constantly extract malware, submit samples, and retrieve results is much more taxing than in the detection process. Overall system wait time is estimated at a little over 2 hours. However, user interaction time is staggering at nearly 5 hours, with 35 separate interaction points, only minutes apart from each other. Not only is this level of required interaction tiresome and frustrating for the user performing the work, it offers many opportunities for human error. The sequential nature of the process represents a system that could be optimized through concurrent activities.

**Private Analysis**

As mentioned in the Prior Research section, the current process for malware detection analysis relies on many third party applications and services. In theory this could be avoided by only utilizing local, homegrown software for detection and analysis. Practically, the Malware Research Team does not have the resources for this, and so dependence on third party systems is necessary. The risk of incorrect results from third party applications is mitigated by utilizing multiple sources, which helps to reduce false positives and false negatives. However, the risk of
confidential information exposure to third parties can only be addressed through the use of localized, private tools.

For the malware analysis process, the MRT currently relies on online services such as CWsandbox, Norman Malware Analyzer, and ThreatExpert. However, a few projects exist which aim to provide practical and accurate malware analysis on a local scale. One of the original projects for malware analysis is known as Truman (Stewart, 2009). It is known as the “reusable unknown malware analysis net.” The premise is that using two computers (one controller and one drone), a known disk image is loaded onto the drone and populated with a single piece of target malware. The drone contains software for recording how the operating system environment changes (due to the actions of the malware), and those changes are reported back to the controller.

Truman, being an open-source project, has been adopted and expanded upon to include additional functionality. One implementation is “Building an Automated Behavioral Malware Analysis Environment using Open Source Software,” by Jim Clausing (2009). Clausing’s employment with AT&T meant that when he discovered malware local to their environment, he did not have the ability or right to submit that malware to online analysis services. This predicament is very similar to the one the Malware Research Team has. Developing a private analysis service such as the one described in the Clausing paper is the best way to preserve the integrity and privacy of malware samples, while still receiving a detailed analysis of a malware’s behavior.

Additionally, the MRT has found recently that the behavior reports from many of the online analysis services are lacking in detail that they used to provide. Clearly, for such a job as determining the exact behavior of malware, excruciating details are important. Pieces of information such as attempted network connections, file modifications, registry modifications, and even individual Windows API calls are all pieces to the puzzle that can determine what a piece of malware is attempting to do. Furthermore, a private analysis service based on open source software has many more options for integration and automation than a third party service, and these options will not be subject to change on another company’s whim. The private analysis service will be a crucial portion of the analysis phase of the MRT’s malware detection and analysis process.
Methodology

The main deliverable for this directed project is a systems design detailing a proposed system for improving, automating, and solving existing problems within the current malware analysis system. A formal systems development methodology will be followed, consisting primarily of a document detailing the architecture, systems context, and technical requirements of the proposed system. Personal communications with Jeffrey Brewer (Purdue University, Fall 2009) have formed the basis for understanding the “state of the art” of systems development strategies. Use cases will provide evidence of the various interactions that users need to have with the system, and will shape the desired state of the system after completion. It is important to note that actual software will NOT be part of the deliverable for this directed project.

In systems design, and particularly software design, a common methodology for the development of a new system is the Systems Development Life Cycle, or SDLC (Satzinger, et al, 2002). The SDLC contains the following phases of systems development:

- Planning – determine the purpose of the system.
- Analysis – determine what the system needs to do, the goals for the system and how to determine if those goals have been met.
- Design – determine how the system will work, what the overall architecture is, and determine what steps would need to be taken to construct an actual system.
- Implementation – using the existing design, construct a system to meet the requirements of the project.
- Testing – establish that the constructed system actually does meet the requirements detailed in the design.
- Maintenance – fix bugs in the system, which are essentially differences between the design (requirements) and the constructed system (reality). As the design inevitably changes, update the actual system to match these changes.

As mentioned, this project is not a software construction project, but instead is a software design project. The SDLC will be followed through the first three steps. This document constitutes the “planning” phase of the SDLC, as it is discussing the purpose of the system. It
begins to address the “analysis” phase, as it has discussed gaps between the current system and the desired system (problems and limitations), which will be used to determine if the proposed system has met its goals. The primary deliverable of the directed project will consist of the remainder of the “analysis” phase and the entire “design” phase. The goal is to provide a technically complete design which would be sufficient to guide someone through software construction, without being so specific as to lock a developer into a technology of set of technologies. Essentially, the directed project will provide a roadmap for development of the automated malware detection and analysis system. The final step of the project will be to validate the design against the current-state challenges identified in Table 2 and Table 4.

**Time Action Plan**

![Time Action Plan Diagram]
**Systems Context Diagram**

- **System Administrators** – allowed to add and remove users from the system in order to manage and control access to use it.

- **Malware Researchers** – primary actors in the context of the system, will configure cases for analysis and retrieve final analysis reports when completed.

- **Customers** – drivers and motivation for the system, will provide images to researchers and retrieve final reports when completed.

- **File Store** – external resource (pre-existing to the system) that provides the actual disk images to be used for the analysis process.
Detection and Analysis Process Flow Diagram

FBI

Results
Disk Image, Case Data

Malware Research Team

Case Data

Malware Identification

Disk Image

Malware Reports

Data

Samples

Malware Samples

Final Reports

Data

Malware Analysis

Samples to Analyze
### Use Cases

<table>
<thead>
<tr>
<th>Use Case Name:</th>
<th>Add New Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>High</td>
</tr>
<tr>
<td>Primary Business Actor:</td>
<td>Researcher</td>
</tr>
<tr>
<td>Description:</td>
<td>This use case describes the process of a malware researcher entering and loading a new disk image for analysis with the system.</td>
</tr>
<tr>
<td>Pre-condition:</td>
<td>Target disk image has been copied to file store.</td>
</tr>
<tr>
<td>Typical Course of Events:</td>
<td>1) → Researcher authenticates to analysis system.</td>
</tr>
<tr>
<td></td>
<td>2) ← System validates researcher’s credentials.</td>
</tr>
<tr>
<td></td>
<td>3) → Researcher adds new case file to system.</td>
</tr>
<tr>
<td></td>
<td>4) ← System prompts for case description and location of target disk image.</td>
</tr>
<tr>
<td></td>
<td>5) → Researcher provides case background, supporting information, and location of disk image.</td>
</tr>
<tr>
<td></td>
<td>6) ← System queues pending analysis job.</td>
</tr>
<tr>
<td></td>
<td>7) ← System notifies researcher when analysis is complete.</td>
</tr>
<tr>
<td>Alternate Courses:</td>
<td>2) ← System rejects researcher’s credentials, access is denied.</td>
</tr>
<tr>
<td></td>
<td>6) ← System cannot locate disk image, analysis job is not added.</td>
</tr>
<tr>
<td>Conclusion:</td>
<td>This use case concludes when a researcher has been notified that his or her analysis job is complete.</td>
</tr>
<tr>
<td>Post-condition:</td>
<td>Researcher generates analysis report.</td>
</tr>
</tbody>
</table>
Use Case Name: Monitor Jobs

Priority: Medium

Primary Business Actor: Researcher

Primary System Actor:

Description: This use case describes the process of a malware researcher monitoring in-progress jobs and estimation of completion.

Pre-condition: Analysis has begun on one or more disk images.

Typical Course of Events:

1) → Researcher authenticates to analysis system.

2) ← System validates researcher’s credentials.

3) → Researcher begins job monitoring

4) ← System polls processing nodes for current job execution status.

5) ← System uses job information to calculate a percentage completion estimate.

Alternate Courses:

2) ← System rejects researcher’s credentials, access is denied.

Conclusion: This use case concludes when a researcher is finished monitoring job status.

Post-condition: Researcher examines final analysis report.
<table>
<thead>
<tr>
<th>Use Case Name:</th>
<th>Generate Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>High</td>
</tr>
<tr>
<td>Primary Business Actor:</td>
<td>Researcher</td>
</tr>
<tr>
<td>Primary System Actor:</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>This use case describes the process of a malware researcher generating and receiving a report detailing the results of malware analysis.</td>
</tr>
<tr>
<td>Pre-condition:</td>
<td>Analysis has been completed on the desired disk image and the researcher has been notified.</td>
</tr>
<tr>
<td>Typical Course of Events:</td>
<td>1) → Researcher authenticates to analysis system.</td>
</tr>
<tr>
<td></td>
<td>2) ← System validates researcher’s credentials.</td>
</tr>
<tr>
<td></td>
<td>3) → Researcher selects desired image/case.</td>
</tr>
<tr>
<td></td>
<td>4) → Researcher issues command to retrieve report.</td>
</tr>
<tr>
<td></td>
<td>5) ← System polls analysis records and builds report detailing findings, then delivers it to the researcher.</td>
</tr>
<tr>
<td>Alternate Courses:</td>
<td>2) ← System rejects researcher’s credentials, access is denied.</td>
</tr>
<tr>
<td></td>
<td>4) ← System cannot generate report because analysis has not been completed.</td>
</tr>
<tr>
<td>Conclusion:</td>
<td>This use case concludes when a researcher has received the report for a given analysis.</td>
</tr>
<tr>
<td>Post-condition:</td>
<td>Researcher examines final analysis report.</td>
</tr>
<tr>
<td>Use Case Name:</td>
<td>Add New User</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Priority:</td>
<td>Medium</td>
</tr>
<tr>
<td>Primary Business Actor:</td>
<td>Administrator</td>
</tr>
<tr>
<td>Primary System Actor:</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>This use case describes the process of the system administrator adding a new account to the system.</td>
</tr>
<tr>
<td>Pre-condition:</td>
<td>Desired username and initial password have been selected.</td>
</tr>
<tr>
<td>Typical Course of Events:</td>
<td>1)  (\rightarrow) Administrator authenticates to analysis system.</td>
</tr>
<tr>
<td></td>
<td>2)  (\leftarrow) System validates administrator’s credentials.</td>
</tr>
<tr>
<td></td>
<td>3)  (\rightarrow) Administrator issues command to add new user.</td>
</tr>
<tr>
<td></td>
<td>4)  (\rightarrow) Administrator enters desired username, initial password, and access level (Researcher or Administrator).</td>
</tr>
<tr>
<td></td>
<td>5)  (\leftarrow) System verifies username is not in use.</td>
</tr>
<tr>
<td></td>
<td>6)  (\leftarrow) System verifies password meets requirements.</td>
</tr>
<tr>
<td></td>
<td>7)  (\leftarrow) System creates new account and notifies administrator of completion.</td>
</tr>
<tr>
<td>Alternate Courses:</td>
<td>2)  (\leftarrow) System rejects administrator’s credentials, access is denied.</td>
</tr>
<tr>
<td></td>
<td>5)  (\leftarrow) System cannot create account because username is already used.</td>
</tr>
<tr>
<td></td>
<td>6)  (\leftarrow) System cannot create account because password is too simple.</td>
</tr>
<tr>
<td>Conclusion:</td>
<td>This use case concludes when the new account has been successfully created.</td>
</tr>
<tr>
<td>Post-condition:</td>
<td>Administrator notifies new user of their username and password.</td>
</tr>
<tr>
<td>Use Case Name:</td>
<td>Remove User</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Priority:</td>
<td>Medium</td>
</tr>
<tr>
<td>Primary Business Actor:</td>
<td>Administrator</td>
</tr>
<tr>
<td>Primary System Actor:</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>This use case describes the process of the system administrator removing a user account from the system.</td>
</tr>
<tr>
<td>Pre-condition:</td>
<td>User no longer needs access to the system.</td>
</tr>
<tr>
<td>Typical Course of Events:</td>
<td>1) → Administrator authenticates to analysis system.</td>
</tr>
<tr>
<td></td>
<td>2) ← System validates administrator’s credentials.</td>
</tr>
<tr>
<td></td>
<td>3) → Administrator issues command to remove user.</td>
</tr>
<tr>
<td></td>
<td>4) ← System verifies that administrator wants to remove account.</td>
</tr>
<tr>
<td></td>
<td>5) → Administrator confirms or denies deletion.</td>
</tr>
<tr>
<td></td>
<td>6) ← System removes account.</td>
</tr>
<tr>
<td>Alternate Courses:</td>
<td>2) ← System rejects administrator’s credentials, access is denied.</td>
</tr>
<tr>
<td></td>
<td>6) → System does not remove account.</td>
</tr>
<tr>
<td>Conclusion:</td>
<td>This use case concludes when the account has been removed.</td>
</tr>
<tr>
<td>Post-condition:</td>
<td></td>
</tr>
</tbody>
</table>
System Objectives

Primary Objectives

The malware analysis system has a number of objectives, which were determined by analyzing the shortfalls of the existing process. Many of these shortfalls are described in the Prior Research section, as these limitations are the sole reason for developing a new system.

The primary objective of the new system is to conserve time. This means that given an acceptable input, the least amount of both system (CPU) and user (personnel) time should be used to arrive at the desired output. Conserving system time in terms of overall CPU hours will be difficult, as the problems being solved by the malware analysis system are complex and require significant processing power to complete. However, wall clock time spent on system processing can be reduced by utilizing more powerful equipment and “scaling out,” e.g. having multiple, similar systems work on tasks in parallel. Therefore, scalability is one of the secondary objectives of the system.

Conserving user time comes down to two separate factors. Some portions of the analysis process will always require human interaction and decision making. Primarily this refers to the final stages of the malware analysis process where results are being analyzed for correlation and unusual findings. However, in the current analysis process there are many “boring” stages where a user must interact with the system to start different stages and perform menial set-up tasks to keep the analysis process running. This type of time expense should be eliminated via automation which is another secondary objective of the system.

Another factor affecting scalability is that of independent tasks. For instance, the detection phase of analyzing a malware case consists of processing the same case data with multiple anti-virus or anti-malware tools. Other than the data they are operating on, each anti-virus tool is independent from the others. This indicates an opportunity for these tasks to be completed in parallel. Parallel computation means that at minimum the same amount of compute time will be used to arrive at the solution, but because the work is being done concurrently across multiple computers, the wall clock time required will be reduced. The difference in process flow between sequential and concurrent (parallel) processing is shown below in Figure 6.
The foundation for scalability exists in the present system through the use of virtualization. Virtualization allows multiple, isolated operating systems to run within a single piece of hardware. This allows a more efficient use of hardware and provides a layer of abstraction for malware processing tasks to occur on any computer in a larger cluster. However, at present there is no infrastructure for managing the hardware that does this processing, or for tracking the analysis tasks that the system has processed in the past or will process in the future.

This limitation contributes to the other secondary objective, which is automation. Because the system cannot track its progress on analysis of a particular case, this work must be done manually by the users. Copying disk images between hosts must be done manually. Starting virtual machines and running software to perform the analysis, and collecting and correlating results at the end must all be done by users. Because of the level of effort required for the analysis of any particular case, both user time and system time increase not only in relation to the complexity of the average malware case, but with the total number of malware
cases that need to be analyzed. Therefore, to serve the primary objective of the project, user interaction with the system should be eliminated as much as possible.

**Other Objectives**

At present, during the malware behavior analysis phase, samples are submitted (by users) to online analysis services. This poses a problem for certain types of cases where privacy is paramount. One objective not related to the primary objectives above is to secure all case-related information within the system and prevent it from being submitted to external entities. The solution for this will be to design or implement an existing private analysis service. One example of private analysis that was mentioned in the review of literature was the Truman sandnet.

**Limitations**

Although not directly mentioned, the Malware Research Team has a need for a case management system that will store case data, additional resources from wherever the case originated (FBI case file notes, for instance), results from analysis processes as well as user generated insights and conclusions. While the database that will power the analysis system is a natural location for such a case management system, it is being considered outside the scope of this project. Instead, this project will focus on the “core business” of generating malware reports from submitted disk images.
The items depicted above represent logical components of the system; their physical location is intentionally not described. Due to the desired scalability of the system, it is presumed that these components can and will exist anywhere.

**Workflow**

- A disk image for a new case to be analyzed is copied to the File Store.
- Using the web-based Management Interface, information about the case (including the location of the image) is entered into the Database.
- The Analysis Coordinator serves as the master node for the operation of the analysis system. Upon seeing a new case, it determines which sub-tasks need to be completed.
- Tasks are distributed to the Worker Nodes for processing.
- The Worker Nodes access resources from the File Store as necessary, copying resources like disk images locally for processing.
- Upon completing their respective tasks, the Worker Nodes notify the Analysis Coordinator.
• The Analysis Coordinator performs final cleanup tasks and updates the Database with the results.

**Functional Requirements**

• The system shall use a relational database to store input metadata, in-progress operational data, and completed results.
• The system shall provide a secured user interface for entering new cases into the system, as well as for generating reports about completed cases.
• An administrative interface shall be provided for management of users within the system.
• Analysis of cases should be coordinated by a single machine/entity, which is responsible for dividing workload among one or more “worker” machines.
• Independent tasks within the detection and analysis phases shall be executed concurrently, saving wall clock time and yielding a quicker turnaround time on case results.
• Analysis shall consist of two phases, to be completed sequentially: detection of malware, and behavioral analysis of malware.
• The detection phase should at minimum consist of the following steps. These steps will be completed once for each malware detection tool that is used, and will be able to execute concurrently on multiple worker machines:
  o Copy active case’s disk image from file store to local disk.
  o Locate existing virtual machine on local disk for selected malware detection tool (preconfigured).
  o Reconfigure virtual machine to use copied local disk image.
  o Boot up virtual machine.
  o Execute script within Windows virtual machine that will use automation to begin scanning disk image using chosen malware detection tool.
  o Following completion of malware detection, script should capture results and report them back to the analysis coordinator.
  o If detection phase is completed, remove disk image from local drive.
• The analysis coordinator shall distribute detection phase work among worker machines until all chosen malware detection tools have been used.
• The analysis coordinator shall convert all results into a unified format and store those results in the database.

• Using the stored results, the system shall identify individual pieces of suspected malware from the active case.

• The analysis coordinator shall assign a task to a worker machine which will open the disk image, and extract all pieces of suspected malware to a temporary location.

• The worker machine shall submit all of the suspected malware to the private analysis service (using an intermediate submission frontend).

• The submission frontend shall process each piece of malware with the private analysis service, and upon receiving results they should be stored in the database with the other case information.

• After all pieces of suspected malware have been behaviorally analyzed, the analysis coordinator shall create a final report of all malware found within the disk image and mark the case as “completed.”

• The system shall notify the user who submitted the case that processing is complete.

• The user shall be able to use the management interface to retrieve the final report of located malware.

**Development Strategy**

Put simply, the development strategy for this project should be agile. Rather than focusing on reams of formal documentation describing how the system works, the focus should be on providing a self-documenting code base that can easily be extended by new developers. As the team who works with the malware analysis system will certainly morph and change over time, a lightweight code base that is easily modified will prove far more important than a far reaching set of documentation.

Additionally, as new malware detection tools are released each year and better behavioral analysis services become available, there will be a definite need to revise the system to keep it up to date. A modular, loosely-coupled approach should be taken so that new features and modules can be added onto the system without compromising the system as a whole. Loose coupling
should also be used to enable unit testing and verification of individual modules before they become part of the primary code base.

Although the representative design featured in this document focuses on a particular programming language, there is no reason that other languages should not be able to be used. This means that a focus should be placed on interoperable standards and cross-platform technologies. At the code level, languages with strong object oriented features and concise syntax should be used, so that the focus is on what the code is accomplishing, not how it is doing it. Standard analysis and design patterns, such as those written by the Gang of Four, shall be used properly to make the application more scalable, maintainable, and updatable.

Agile development makes sense for a project of this size because requirements can be prioritized. The first thing to be done is defining the data entities and attributes that the system must track. Secondly, the analysis coordinator should be developed, in a generic fashion so that instead of the specialized tasks it will be performing in this system, it could dispatch tasks of any type to any system. This guarantees that it will be expandable to meet the future needs of the project. Following that, software for the worker machines should be developed. At this point, the system will be capable of tracking and distributing arbitrary tasks among a cluster of machines.

The most difficult part of any software development project is integrating with third parties. In this project, virtualization technology is used to provide a safe, isolated, and scalable environment for both the execution of malware detection tools as well as the actual execution of malware for behavioral analysis. Nearly all virtualization applications have APIs available to reconfigure and execute virtual machines and applications inside of them. The software on worker nodes will be responsible for integrating with these virtual machines and running software, monitoring for faults and collecting results. Again, these components should be loosely-coupled, so that they can be extracted and unit tested for correct behavior before integration into the rest of the software in the project.

This unit testing carries with it an important benefit: due to agile development, one entire development phase can be devoted to the completion of a single component of the system. As long as everything is done to allow that component to work, there may be enough of the project completed to achieve a real benefit (some level of automation) even though the project as a
whole is not completed. During the subsequent development phases as other components are completed and integrated, the system will become better and easier to use. However, smaller benefits occur earlier in the process due to the agile method of “deliver early, deliver often.”

In development of the centralized analysis coordinator as well as the worker machine software, open standards such as SOAP (or similar web services) should be used to avoid lock-in to a given vendor or technology. In the future this would allow changes to operating systems or programming languages with minimal effort. In the same vein, a management interface needs to be provided, which could simply be written to access the malware analysis database directly. Instead, providing an intermediate API would allow different interfaces to the system or even future automation potential with little to no cost to the original developer.

**Reference Implementation**

The preceding pages only detail the design of the desired system in a way that is abstract from technology. However, all technologies required to develop the required components for this system presently exist. Below is a reference implementation showing the physical layout of the system along which components might exist where and what external resources will be used.
The reference implementation primarily uses Windows operating systems running on standard Intel-based x86 hardware. The overall architecture consists of a master node which coordinates analysis across a larger group of worker nodes. Each worker node runs VMWare Server, a virtualization technology that will allow each node to virtually run many more isolated operating systems within.

To begin, an administrator will use the Management Web Site to add a new researcher into the system. Login account information is stored in the Microsoft SQL Server database. Next, a researcher can login to the web site, which will allow him or her to add a new case into the system. The case information is also stored in the SQL database. Any new case added must reference a disk image that exists on the File Store.
Upon seeing that a new case has been added to the database, the Analysis Coordination Service goes into action. This service, written in C#, watches for changes in the database and communicates with the Analysis Gateway on each of the worker nodes, distributing tasks and facilitating the completion of the analysis. The analysis coordinator will have knowledge of each of the worker nodes (configured in the database), and will be able to communicate with them via Windows Communication Foundation, a communications technology that is available to C# and is part of the Microsoft .NET Framework. The advantage of writing the components of the system in C# is the support for Computer-Aided Software Engineering (CASE) tools to be used to reverse engineer the design and provide accurate UML diagrams to supplement design documentation for the reference design.

The Analysis Gateway runs on each worker node. It is responsible for the following tasks:

- Copy disk image from File Store to local disk, for performance reasons.
- Configure the underlying virtualization technology (in this case VMware) using the VMware API. This API will be used to create new virtual machines, start them, and stop them.

As shown in diagram, each worker node runs VMware Server, which provides a base that can host multiple virtual machines (VMs). Four are shown in the diagram, but many more could exist. This ability to run multiple isolated operating systems on a single machine serves to scale the system up, allowing faster hardware to be used for increased capacity as demand for the system rises. Because multiple worker nodes can be used to simultaneous process a case, or to even process altogether different cases, this allows the system to scale out. More worker nodes can be purchased and added to the system as it grows.

Each worker node will have two “template” VMs on it. One will be for the detection phase of the analysis process, and the other will be for the behavioral analysis phase. As shown, these two VMs are considerably different, as one is Windows-based and the other is Linux-based. The common factor is that some piece of common code is placed on the machine that will allow it to contact the Analysis Coordinator, receive tasks, and perform actions on the virtual system. For the detection VM, this is the Detection Monitor, while for the analysis VM, it is the Analysis Monitor. Linux has increasingly good support for C# applications, which makes it an acceptable choice for cross-platform use.
After the Analysis Gateway copies the disk image locally, it can create an instance of either the detection or analysis VMs, and configure that VM to reference the local disk image. After starting the VM and allowing it to “boot” up, the appropriate Monitor inside the VM will take over and communicate with the Analysis Coordinator directly. The Monitor will begin execution of the desired detection or analysis software (McAfee, et al for detection, Truman for analysis). Upon completion of the desired task, results are gathered and returned to the Analysis Coordinator, who can then signal the Analysis Gateway to shut down the VM and release any consumed resources.

**Recommendations for Future Research**

It should be clear that there is an opportunity for future research beyond the design presented in this paper. This paper describes only a design and not the results of an actual implementation. An excellent opportunity for future research would be to utilize the design presented here (and possibly even the reference implementation) and actually develop code to fulfill the roles described.

**Conclusions**

The goals of this project were to overcome limitations of the existing malware analysis system. Limitations of the existing system included wasted time, and a lack of scalability and automation. The new system is designed to meet the objectives outlined in this document, and to eliminate the limitations listed above. Although it would have been prohibitively difficult to design such a system as well as implement it, the design presented here should provide a solid foundation for future work. Additionally, it should serve as future evidence for gathering limitations and requirements about a real world problem and designing a solution to meet those requirements.
References


