Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualizing the Malicious Threat Landscape

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Florian Lukavsky
Matrikelnummer 0325558

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Priv.-Doz. Dipl.-Ing. Dr. Engin Kirda
Mitwirkung: Univ.-Ass. Dr. Paolo Milani Comparetti

Wien, 01.05.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Abstract

Today, malware is the biggest security threat on the internet. Antivirus software vendors face thousands of new viruses on a daily basis. These data sets can hardly be managed manually. Therefore, dynamic and automated analysis systems are used. Anubis [7] is such an analysis system. It executes binaries in a controlled environment and generates reports, describing the actions performed by this submitted binary. Anubis is actively deployed since the beginning of 2007 and has already analyzed more than two million malware samples since then. But the automated analysis leads to another problem. There are simply too many reports to get a complete picture of the behavior of malware. This calls for one more layer of abstraction, a high-level statistical analysis to identify trends in malware behavior.

In this thesis we present a database schema that allows us to store analysis reports by Anubis efficiently. This is the prerequisite for queries on this data to gain detailed insight into the behavior of malware. We develop an extensible framework for the creation of high level views of malicious behavior. These statistics are updated on a daily basis, which allows a near real-time view on current malicious behavior. This aids the discovery of trends when observing malicious behavior over a longer period of time and favors the identification of correlations between different behavioral characteristics.

In order to detect interesting high-level behavioral characteristics, we have to map these to the corresponding low level events captured by Anubis. We detect behavioral characteristics like the installation of Browser Helper Objects for the Internet Explorer or the alterations of security settings of the Windows Firewall. Altogether, we observe 31 behavioral characteristics of the categories ‘file activity’, ‘network activity’, ‘registry activity’ and ‘system interaction’. In addition, we generate statistics that represent the state of Anubis and describe the quality of submitted samples.

Kurzfassung

Heutzutage stellt Schadsoftware die größte Bedrohung im Internet dar. Die Hersteller von Antivirensoftware sehen sich täglich mit tausenden neuen Viren konfrontiert. Diese Datenmengen sind manuell kaum noch zu bewältigen. Daher kommen dynamische und automatisierte Analysesysteme zum Einsatz. Anubis [7] ist ein solches Analysesystem und beobachtet das Verhalten eines zu untersuchenden Programms und generiert einen Bericht dieser Beobachtungen. Seit der Lancierung von Anubis Anfang 2007 wurden mehr als zwei Millionen Programme analysiert. Aber die automatisierte Analyse führt nun zu einem weiteren Problem. Es gibt zu viele Analyseresultate um das Verhalten von Schadsoftware im Ganzen zu betrachten. Eine weitere Schicht der Abstraktion ist notwendig, eine statistische Analyse von high-level Verhalten, die Trends des Verhaltens von Schadsoftware aufzeigt.

In dieser Diplomarbeit präsentieren wir ein Datenbankschema, das es uns erlaubt die Berichte von Anubis effizient zu speichern um in weiterer Folge mittels Abfragen auf diesen Daten detaillierte Einblicke in das Verhalten von Schadsoftware zu erhalten. Dafür entwickeln wir ein Framework zur Erzeugung von Statistiken über bösartiges Verhalten. Diese Statistiken werden täglich aktualisiert, was nahezu eine Echtzeitsicht auf das Verhalten zulässt. Dadurch wird es ermöglicht, Trends des bösartigen Verhaltens von Viren und Schadsoftware abzulesen und Korrelationen zwischen unterschiedlichen Verhaltensmustern zu identifizieren.

Um bösartiges Verhalten festzustellen, identifizieren wir Verhaltensmerkmale wie beispielsweise die Installation von Browser Helper Objects für den Internet Explorer oder das Ändern von Sicherheitseinstellungen der Windows Firewall. Dieses abstrahierte Verhalten müssen wir auf low-level Events, die von Anubis aufgezeichnet werden, zurückführen. Wir beobachten nun 31 Verhaltensmerkmale aus den Kategorien “Dateisystemaktivität”, “Netzwerkaktivität”, “Registryaktivtität” und “Systemwechselwirkungen”. Zusätzlich generieren wir noch Statistiken, die den Zustand von Anubis beschreiben und die Qualität der eingeschickten Dateien widerspiegeln.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Today, malware is the biggest security threat on the internet. Antivirus software vendors face thousands of new viruses on a daily basis. These data sets can hardly be managed manually. Therefore, automated dynamic analysis systems are used to gain understanding of the actions taken by malware in order to develop countermeasures. Anubis [7] is such an analysis system. It started off as a command line tool that executes and monitors binaries in an unmodified Windows XP environment and evolved into a large-scale dynamic analysis system.

Anubis is actively deployed since the beginning of 2007 and has already analyzed over two million malware samples since then. Anubis receives most of these samples, almost 90 percent, through feeds from security organizations and antivirus software vendors like IKARUS Security Software GmbH. Furthermore, Anubis allows automatic submission from honeypot deployments, by providing an URL that is compatible with the submit handler of nepenthes [19]. Additionally, samples can be submitted manually via a public web interface, but samples collected through this channel represent only a fraction of the total number of submitted samples. We receive a very diverse mix of current malware samples from almost 900 registered users and over 110 000 unique IPs.

Even though the analysis report that is generated by Anubis for each analysis presents a concise overview of the behavior of the sample, the sheer number of reports generated by Anubis prohibits a manual analysis of each analysis result. In order to nonetheless gain an overall understanding of the behavior exhibited by malware, we introduce a number of statistics on the observed behavior, and provide interactive visualization tools that allow an analyst to explore these statistics. Our goal is to have a near real-time view on current malicious behavior that allow us to see trends in malicious behavior over a longer period of time and the ability to identify correlations of different malware behaviors. To this end, we identify interesting high-level behavioral characteristics of malware, like autostart behavior or the alteration of security settings. In order to extract these characteristics we have to understand all the different techniques used by malware to accomplish such goals. Then, we have to investigate how they map to Windows native system calls and Windows API functions captured and stored in the analysis report by Anubis.

A spam bot, for example, will likely scan the compromised system for email addresses, in addition to its primary task, the sending of emails. We can conclude that a sample harvests email addresses if it reads email folders of Outlook Express or the Windows address book.

Spy ware and ad-ware often make use of Browser Helper Objects (BHO) to control the Internet Explorer and monitor user's actions on the internet. In order to install a BHO, a sample needs to register its COM object in the Windows registry. A modification in this registry key reveals this behavior.

Furthermore, we require additional visualizations that illustrate the state of health of Anubis and depict the quality of submitted samples or the number of analyses that have been performed by Anubis.

The generation of visualizations on malware behavior requires extensive and heavy queries on the entries of a potentially high number of analysis reports. In order to have a near real-time view on malware behavior, these visualizations have to be generated in a timely manner. The XML files of the original analysis report do not meet this requirement very well. Therefore, we analyzed the current database design to reveal potential in performance and efficiency gains and extended Anubis' relational database to store the data of the analysis reports efficiently. We exploit properties of the analysis report to avoid redundancies and thus save storage space. After the analysis of a sample is completed, the content of the analysis report will be parsed into the database and is then available in a format that favors efficient queries. In order to be prepared for future trends in malware behavior, we developed an extensible and adaptable query framework to generate high-level views of malicious behavior on a daily basis. Currently, we maintain 42 statistics covering all kinds of behaviors like autostart behavior, network traffic or modifications to the file system.

Visualizing these statistics requires web-based tools that do not only display the data, but allow interaction with the data as well. We evaluate such tools and frameworks, paying special attention to the requirements of Anubis and the large datasets they must be able to cope with. We conceive strategies on statistics generation and investigate different forms of their visualization. To this end, we discuss design decisions and document our implementation.

The generated statistics give us detailed insight into current malware behavior. Not only were we able to confirm obvious correlations between different malware behaviors, like that a malware sample that harvests email addresses frequently initiates SMTP connections, but we found, for example, that IRC bots commonly alter the security settings of the Windows firewall and edit the Windows hosts file [1]. We were also able to detect trends in malware behavior. To ensure that the malware is executed again upon system reboot, installing a Windows service declines in popularity while other techniques to ensure automated execution gain in popularity. Furthermore, by identifying trends in malware behavior, we can find weak points of the Anubis analysis process, which set the direction for future improvements. Additionally, the statistics on the state of health of Anubis, in particular, allow us to quickly identify bugs and take according counter measurements. For example, we found that we did not classify the file type of all submitted samples correctly.

[1] The Windows hosts file maintains domain name to IP mappings and overrules the results of a DNS server

1.1 Overview

In Chapter 2, we discuss state-of-the-art work that is related to our work in a number of ways. We describe previous approaches to visualize malware behavior and trends of this behavior. We contrast our approach that depicts behavior of multiple malware samples to the visualization of a single malware sample and point out how the behavior of malware can be exploited for the visualization in contrast to representing an arbitrary binary file. Finally, we describe visualizations in related domains and how they are used to depict malicious behavior in intrusion detection systems and to illustrate network attacks.

In Chapter 3, we review the structure of the analysis report that is produced by Anubis for each analysis of a sample. We discuss multiple options to efficiently store and access the data of the analysis reports. In particular, we list advantages and disadvantages of XML databases, multidimensional databases and relational databases. We describe the schema that we designed in order to ensure efficient storage of the behavioral data and the indexes that are in use to speed up the queries and allow for quick access to the data. To this end, we document the process of parsing the analysis report into the database and specify a set of analysis reports that we used to test and validate the parsing process.

In Chapter 4, we discuss general aims of visualizations and present an overview on human perception. We distinguish preattentive and attentive processing in order to point out the importance of preattentive features for clear and understandable visualizations. Furthermore, we investigate methods for the visualization of time-dependent data and for the visualization of proportional data. We point out the importance of interactivity in visualizations of time-dependent data. Finally, we summarize our findings and formulate demands on visualization tools. We use these demands to evaluate tools in order to ensure that they fulfill our requirements.

In Chapter 5, we depict the design of the distributed system that is responsible for generating and displaying the statistics on malware behavior. We discuss design decisions and document the implementation of the generation component and the component that is responsible for the display of the visualizations, as well as the data exchange between these components. In particular, we point out how to extend this system and include new queries in order to generate new views on malware behavior.

In Chapter 6, we present different views into malware behavior. We start off by describing the interaction between user and charts and by listing how these views are categorized. Furthermore, we describe each chart and point out the origin of the data for each graph in a chart. For these charts, we discuss the gained insights, trends of the represented behavior and correlations between different malware behaviors. In order to support our conclusions, we give examples that provide insight into malware behavior and present additional facts that are related to these behaviors.

In Chapter 7, we discuss future work regarding the statistics generation environment and Anubis in general. We propose to keep an eye out for interactive statistics tools that do not rely on proprietary browser plug-ins but make use of the HTML5 standard instead. We point out that the current set of views on malware behavior gives a good coverage, but is by no means complete. We propose additional views on malware behavior that would require adaptations to the Anubis environment and the generation of the analysis reports. Additionally, we argue how modifications to the analysis report could lead to a more efficient and more effective query environment.

CHAPTER 2

Related Work

2.1 Malware Behavior

Many aspects of malware behavior can be examined. The process of malware propagation and the interaction between the infected system and other hosts on a network are the best understood aspects. But in this work we want to concentrate on interaction of malware with the infected system.

- **FIRE** [14] is a system that identifies rogue networks and internet service providers. It relies on Anubis [7] and Wepawet [22] as data sources and tries to identify command and control servers, phishing hosting providers and drive-by-download hosting providers [66]. The number of rogue servers existing in an autonomous system along with their uptime is used to compute a malscore that defines the maliciousness of an autonomous system. Furthermore, these results are published daily as a list of IP blocks.

  For each autonomous system, statistics are offered, covering the past six months. On a daily basis, these statistics describe the course of its malscore, its ranking compared to other autonomous systems and the number of malware servers in this system. In addition, a world map visualizes the geographical location of current rogue networks and servers.

- Since Google is interested in presenting high quality search results, they put a lot of effort into analyzing and identifying malicious web sites. If a URL listed as part of a search result is found to be malicious, the user is warned and the link is marked as malicious. Furthermore, Google allows applications to access these URL blacklists via the **Google Safe Browsing API** [15]. Given their objectives, it is clear that Google is mostly interested in the behavior of the infection process itself, i.e. distribution of malicious web sites and the propagation of malware to end user systems, rather than the behavior of the malware once it has infected the target system [59].

  The malware itself is mostly hosted on a few exploit servers while the actual process of infecting end user systems is initiated via compromised legit web sites [60]. Usually, the

user's system is tested for vulnerabilities to run exploits that initialize so-called drive-by-downloads, thus executing arbitrary code on the user's system. Google detects this behavior by executing suspicious web sites in a virtual machine and comparing the number of running processes before and after the execution. In [59, 60] they visualize the results in several charts, they compare the number of malicious with harmless URLs, they measure the distribution of malware categories and the lifetime of malware distribution hosts.

In later work, the authors extended the analysis process to allow detailed analysis of the malware's network activity [56]. With this extension, it is possible to provide detailed statistics on the distribution of utilized network protocols and destination hosts and ports. The analysis of a malware's network activity allows drawing conclusions about the goals of the malware. For example, one can often see malware that reports home harvested data or that requests additional executables and instructions.

- In [30], Bayer et al. give a thorough overview on current malware behavior. For their paper he generated statistics on all submissions to Anubis between the launch of Anubis in February 2007 and end of 2008. In a first step, they visualize the throughput of Anubis, and compare the number of submissions with the number of analyses and unique analyses and list the number of different submitters along with the amount of their submissions. The analysis of malware behavior starts with the differentiation of submitted file types. PE-Files are examined for their packers before the actual behavior of the samples is looked at. Behavioral features include file system activity, registry activity and network activity. Most of the behavioral features are listed in tables along with the percentage of their occurrence per sample and per malware family over the whole observation period and lack a more fine-grained perspective. Visualizations are only used for network traffic analysis and represent the chronological sequence of network events.

  In this thesis, we aim to extend the work done in [30]. We desire a more fine-grained perspective on all these behavioral features. Furthermore, we require a temporal dependence to reflect the change of malware behavior over time. While data gathering in preparation for [30] was not designed for continuous update of these statistics, constant and regular updates ought to be a key feature of the future statistics framework.

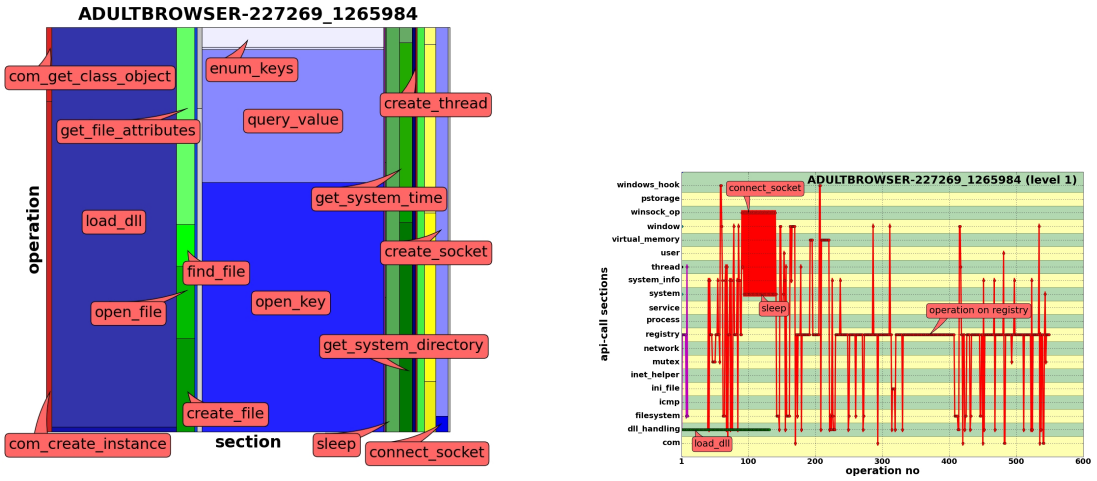
2.2 Malware Visualization

Visualization of malware can be accomplished by potentially every tool or system that is capable of visualizing binary executables. Visualization can either be based on an executable's behavioral profile resulting from a dynamic analysis or the results of its static analysis.

Generally these approaches try to support some kind of visual clustering of malware. The analyst is given the possibility to rapidly find similar executables by comparing their visual representation rather than by comparing a possibly very large list of properties.

- **CWSandbox** [11] proposes two approaches for visualization of malware. Both approaches rely on a behavior report generated by CWSandbox during the dynamic analysis of the malware. Initially, this report contains data that is too fine-grained for visualization.

Therefore, an abstraction of this report is produced. The abstraction contains the API calls executed by the sample along with the API call's most significant parameter [67]. They propose two approaches for generating this abstraction.



(a) Tree map of the Adultbrowser malware.

(b) Thread graph of the Adultbrowser malware.

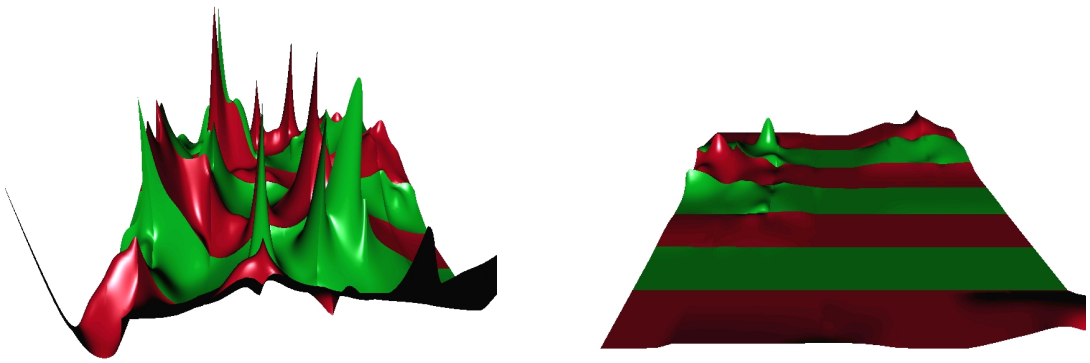
Figure 2.1: Visualizations of malware in CWSandbox. [67]

The first approach, represented in Figure 2.1a, generates a tree map. The 120 monitored API calls are separated into 20 distinct sections. Each section is assigned a differently colored rectangle. The size of the rectangle depends on the number of recorded API calls.

The other approach, illustrated in Figure 2.1b, generates a thread graph. The x-axis represents the temporal dimension while the y-axis describes the behavioral dimension. This visualization does not only give an overview of the API calls executed during the analysis, but it also allows the construction of a causal relationship to the time of occurrence and the actual process.

- Panas et al. propose a visualization of binaries as a three dimensional landscape [55]. In a first step, they parse the binary as an abstract syntax tree. In the next step, they calculate metrics based on the number and time of control transfer instructions, instructions within a function, and data transfer instructions. Each of these metrics is mapped on one of the three axes. The x-axis and the z-axis are normalized to a fixed size and therefore these metrics do not influence the area of the landscape. The result is a unique mountain range that represents the signature of a binary. Figure 2.2a represents the signature visualization of 'Klez c' malware. To conclude the similarity of two binaries from these images, their signature visualizations can be subtracted, which produces a so called delta signature visualization. The flatter the resulting landscape the more similarities can be conducted, which can be seen in the delta signature of 'Klez a' and 'Klez c' in Figure 2.2b.

- **VERA** is a framework that visualizes the program flow of an executable. By monitoring the executable in a modified XEN image, it identifies basic blocks and represents them as nodes.



(a) Signature visualization of Klez c malware.

(b) Delta signature visualization of Klez a and Klez c malware.

Figure 2.2: Visualizations of malware signatures. [55]

Transitions between these blocks are represented as edges and the number of transitions determines the thickness of the edges. Figure 2.3 gives an example of the visualization of the Netbull virus protected with the Mew packer. VERA's main purpose is to aid in reverse engineering of compiled binaries, possibly obfuscated by packers or compressed. The visualization helps to find the entry point of an executable and to understand the basic functionality of the analysis subject [61].

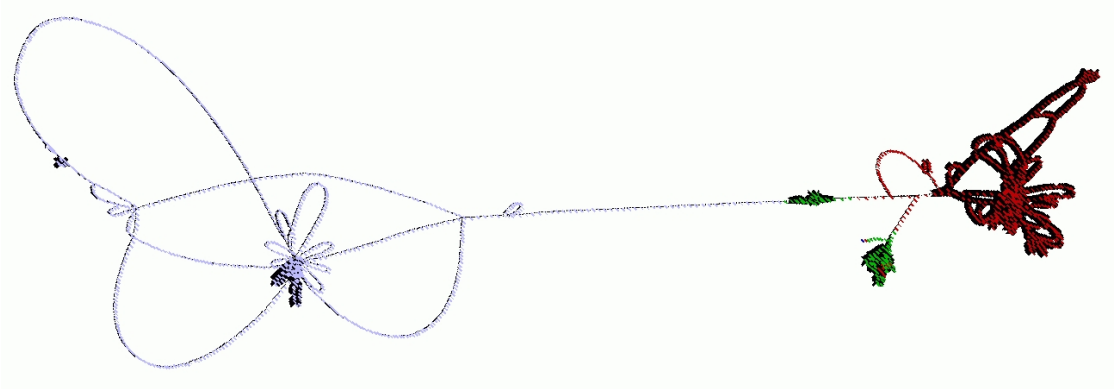


Figure 2.3: Visualization of the Netbull virus protected with the Mew packer. [61]

- **Malwarez** uses API calls, memory addresses and subroutines of a disassembled malware sample and their kind of appearance as input parameters for an algorithm that grows an artificial three-dimensional organism [33]. Figure 2.4 shows the visualization of the MyDoom malware. This system is not intended for malware analysis, but its purpose is to generate art.

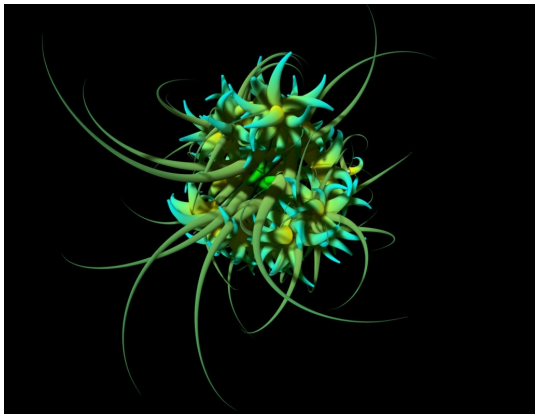


Figure 2.4: Visualization of the MyDoom malware. [33]

2.3 Visualization of Files

Visualization based on the plain binary data of a file is possible as well and algorithms for this task can be found in [8]. But the results do not say anything about the behavior of an executable binary or the similarity of the behavior of two binaries. Two binary files could be very similar while doing something completely different. Furthermore, they would not necessarily have to be executables. A video or music file could serve as source just as well. Or they could result in very different visualizations although they might be performing the same task but are merely differently packed.

It seems that the only use of this approach is to generate art or gimmicks. We have found no source that seriously proposes such an approach for malware visualization as a step of malware analysis.

2.4 Visualization of Network Attacks and Intrusion Detection

Network attacks happen to be very complex because they involve a large number of hosts and an even larger number of network connections. To understand the process of a network attack, many log files have to be examined. For a human, this is a time-consuming and cumbersome task. This is why researchers tried to find ways to visualize such attacks. A few images can contain all the valuable information about a network attack, thus making examination of all log files superfluous. The analyst gets a clear idea about the happening in a few seconds. This allows analysts and system administrators to react to attacks more quickly and more efficiently and to detect intrusion attempts early.

Because of the complexity of network attacks, many different approaches of visualization have been proposed.

- The most popular approach is the visualization as a graph. Each node represents a host and each edge represents a network connection. In [53] and [48] this approach is taken one step further and the graphs are combined with several other coordinated visualizations like

tree maps. This aids the process of drilling down to the most interesting parts of the graph and filtering irrelevant information.

- **VisFlowConnect** takes a slightly different approach. The parallel axis representation is used to visualize incoming and outgoing connections [72]. The external sender, the host and the external receiver are each plotted on one axis. A link between these axes represents the network connection. A replay function is available to examine temporal changes.

- Abdullah proposes simple stacked bar chart histograms as visualization for his intrusion detection system [25]. The traffic for a given time frame is recorded. The size of the bar for a given time frame represents the amount of connections that occurred in that period. The port ranges are highlighted in different colors, thus making it an easy task to identify sudden abnormalities. To increase the clarity of the charts, the user is given the option to filter the port ranges and adjust the time frames.

- **VisAlert** proposes a novel visualization paradigm. Instead of visualizing the raw network traffic like the previous approaches, VisAlert starts one abstraction level higher and visualizes alerts produced by intrusion detection systems [47]. The 'where' dimension is mapped to a point in a circle, thus allowing for representation of the network topology. The 'what' and 'when' dimensions shape a ring around the 'where' dimension. Each alert can now be visualized as a line between the 'where' domain and the 'what' and 'when' domain. Hosts that produce many alert messages can now be easily identified, by the number of adjacent lines. An example of a visualization of a network attack is given in Figure 2.5.

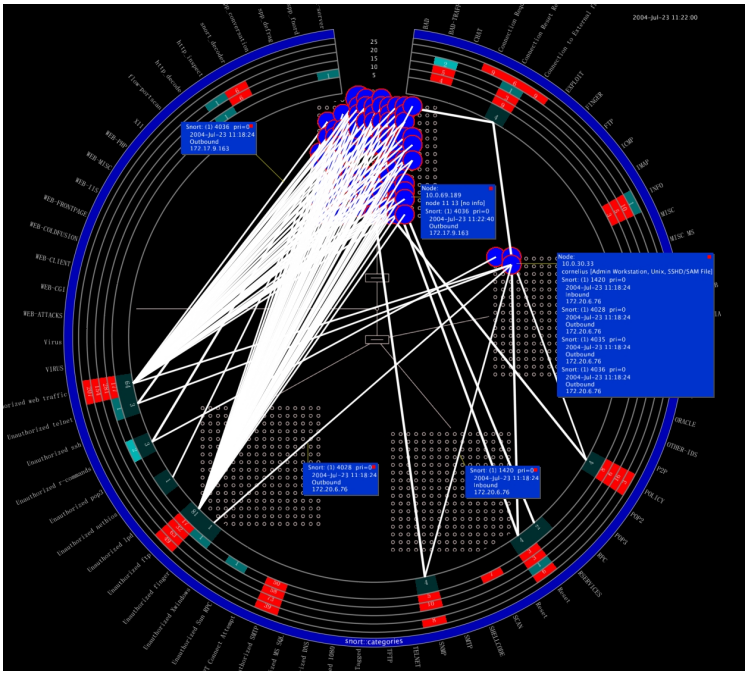


Figure 2.5: Visualization of a network attack in VisAlert. [47]

The visualizations of network intrusions and network attacks are very diverse. We have seen simple graphs, tree-maps or novel paradigms that handle multiple dimensions. Other more exotic approaches use concepts of electromagnetism, fluid dynamics and gravitational theory to visualize the data of network intrusions [65]. But all these approaches have one thing in common: They allow the user to interact with the visualization rather than to present a rigid image of the same. This favors a better examination of suspicious events and helps in better understanding the overall picture.

CHAPTER 3

Efficiently Storing the Analysis Data

For each analysis of a file, Anubis produces a XML report that describes the behavior of the file at a high level. In particular, it contains the Windows native system calls and Windows API functions that are called during analysis along with their parameters. In total, Anubis monitors 35 system calls and API functions that are categorized in the sections “file activities”, “registry activities”, “service activities”, “process activities”, and “miscellaneous activities”. In addition to these activities, Anubis parses the network dump that has been generated by the sample during analysis and extracts information on the most popular network protocols like HTTP, SMTP, FTP, IRC, and ICMP into the category “network activities”. To draw conclusions about common behavior and to generate statistics that take into account the behavior of many files, it is important to store the large number of behavioral reports in a way that allows for efficient querying of the data. In this chapter, we will discuss several approaches to reach this goal and describe the approach that we chose to accomplish this task.

3.1 Storage Engine

XML Databases

Since all the reports, produced by Anubis, are stored as XML files [1] it seems reasonable at first glance to access the reports and their content via a XML database. A XML database is simply a repository, where queries can be executed on the available data and new XML documents can be inserted.

Although this approach does not require any further data transformation and data cleansing, XML databases come with obstacles. Due to XML’s nature to store the metadata along with the data in the same document, a drastic increase of the document’s size is inevitable. Recently, some approaches have been presented to counteract this draw back. Compressors like XMill [46]

[1] Although Anubis currently uses a database to manage submissions and handle report requests, the core analysis component of Anubis is designed to work in an environment without any database back-end.

exploit XML's characteristics and others like XQueC [29] go a step further and optimize the compressed XML documents for queries. With currently more than 3 million completed analyses compression of the generated reports is an absolute necessity. At present, these XML reports are compressed using *gzip*, which reduces their overall size to 18 GB. However, compression causes an overhead not merely during initial storage, but during every query on the document. While the compression cost currently hardly makes a difference, since a document is typically read a few times only during its lifetime, it has huge impacts when it comes to generating statistics. A document might be queried multiple times daily to fulfill this demand.

For daily queries, performance becomes a major issue. In general, a relational database outperforms any XML based database when it comes to queries [71]. This ensues mainly from the fact that parsing of each XML document is required to extract the desired data. Although parsing can be parallelized, [52] states that any delimited format performs orders of magnitudes better than XML.

Powell states that native XML databases are most appropriate for "small scale applications, small quantities of data, very few users (low concurrency and multi-user requirements), low security, low data integrity needs, and above all, low performance" [58]. Anubis' data storage is not intended solely for statistics generation, but its data will be made available to other applications, for example to the WOMBAT API[24]. Therefore, large scale applications cannot be ruled out. We have already identified that data quantities will not be small. Furthermore, with at least 3000 analyzed samples daily, the XML database would grow by at least 75 MB a day. Security, concurrency and data integrity are no major concerns for the data storage, but, like stated previously, performance will play a leading role.

For the reasons stated above, we determined that XML databases do not meet our requirements.

Data Warehouses and Multidimensional Databases

The preeminent tasks of a data warehouse are to load data and to access and analyze the data once it is integrated into the data warehouse. Usually, the data is not updated after it is loaded into the data warehouse. Therefore, data warehouses do not require record-level and transaction-based updates and can go without techniques like row-level locking, commits, logging and checkpoints, which would introduce an unnecessary overhead when inserting or accessing data [45].

The objectives of a data warehouse meet our requirements for an information delivery system to a great part. Once the data is loaded we do not want to update it anymore. Additionally, like in a data warehouse, we want to access and analyze the data efficiently.

In order to analyze the data, data warehouses draw upon multidimensional databases [62]. Multidimensional databases save the data in hyper-cubes and use slicing and dicing techniques to access the desired data. In order to store the data as hyper-cubes, the data needs to be made available in a star-schema [57]. In a star-schema, a central fact-table holds references to all the dimensions for each entry.

Unfortunately, it is not feasible to store the data of the analysis reports in a single hyper-cube. Of course, we could define each of the categories of activities as a dimension and store the simple occurrence of an activity in the fact table. However, utilizing this approach would cause precious

information to be lost. We could only make statements on the existence of activities, but we would use the information on the details of the actions performed.

Instead, we would have to create hyper-cubes for each activity, which would share some dimensions like the date of the analysis and the analysis subject. The details of each activity would form the remaining dimensions. This approach would make perfect sense if our main focus would lie on the analysis of each activity separately. However, we want to be able to place more thorough queries that involve multiple activities. This would be a cumbersome task when utilizing multidimensional databases, as multidimensional databases are optimized for joins on a predefined set of dimensions.

While multidimensional databases do not cover our needs, we can pick up many of the design principles for data warehouses for the design for our data storage. Utilizing the star-schema for the storage of the activities will speed up queries in relational databases as well.

Relational Databases

Relational databases have been studied for more than 40 years and they are optimized for the storage and retrieval of huge data sets. In a relational database, the data is organized in two-dimensional tables [54]. By joining multiple tables, complex views on the data can be generated. Selection and projection allow filtering the data only for relevant entries.

Currently, we operate a relational MySQL database to manage the submissions to Anubis and to maintain meta-data on the analyses. But we do not store any behavioral data, which is available in the analysis report, in the database itself. It seems natural to extend the current database design to meet the new requirements.

While discussing data warehouses and multidimensional databases, we concluded that mechanisms of relational databases like locking, commits and logging lead to an overhead [64], even when we do not require these features. A simple relational database that does not support transactions seems perfect in order to meet our requirements.

MySQL offers a storage engine, MyISAM, which does not support transactions [64]. Since we do not require transactions for data access and the simple insertion of new data, we avoid a huge overhead by abstaining from this feature of relational databases. To insert additional rows, no locking is required. Thus, insertion of new data and the generation of statistics can take place concurrently.

With these features of MySQL we are able to optimize a relational database to our needs, which will allow the effective storage of the behavioral data and favor efficient queries. The fact that we employ a relational MySQL database for Anubis already is an additional advantage of this decision, as we can simply extend the existing database and do not have to introduce another technology, which would add even more complexity to the environment of Anubis.

3.2 Schema

As a preparation to the schema creation we have to identify the information inside the XML report that will give us insight into malware behavior and is therefore required for the statistics generation. Some meta-information about the analysis itself is maintained in the database already.

The meta-information includes the date of the analysis, the version of Anubis that was used for the analysis and the time that was required to analyze the sample. Since this information is available in the database already, we can refer to it and do not have to store it separately.

Next, we need to resolve the level of detail of the behavioral data that we require in the database for the generation of statistics. In the analysis report, all activities are linked to each analyzed process individually. If the primary analysis subject creates a new process, the actions performed by the new process will be stored separately. For the extraction of a malware's behavior we are not interested in the particular process that triggers a certain action and therefore do not require this fine-grained level of detail. Instead, it will be sufficient to link an action that was monitored at some point during the analysis to the analysis itself. This decision reduces the complexity of the behavioral data that is stored in the database, as we ignore the actual process that is responsible for an action, but we link all the actions of the entire report only to a unique ID for the sample.

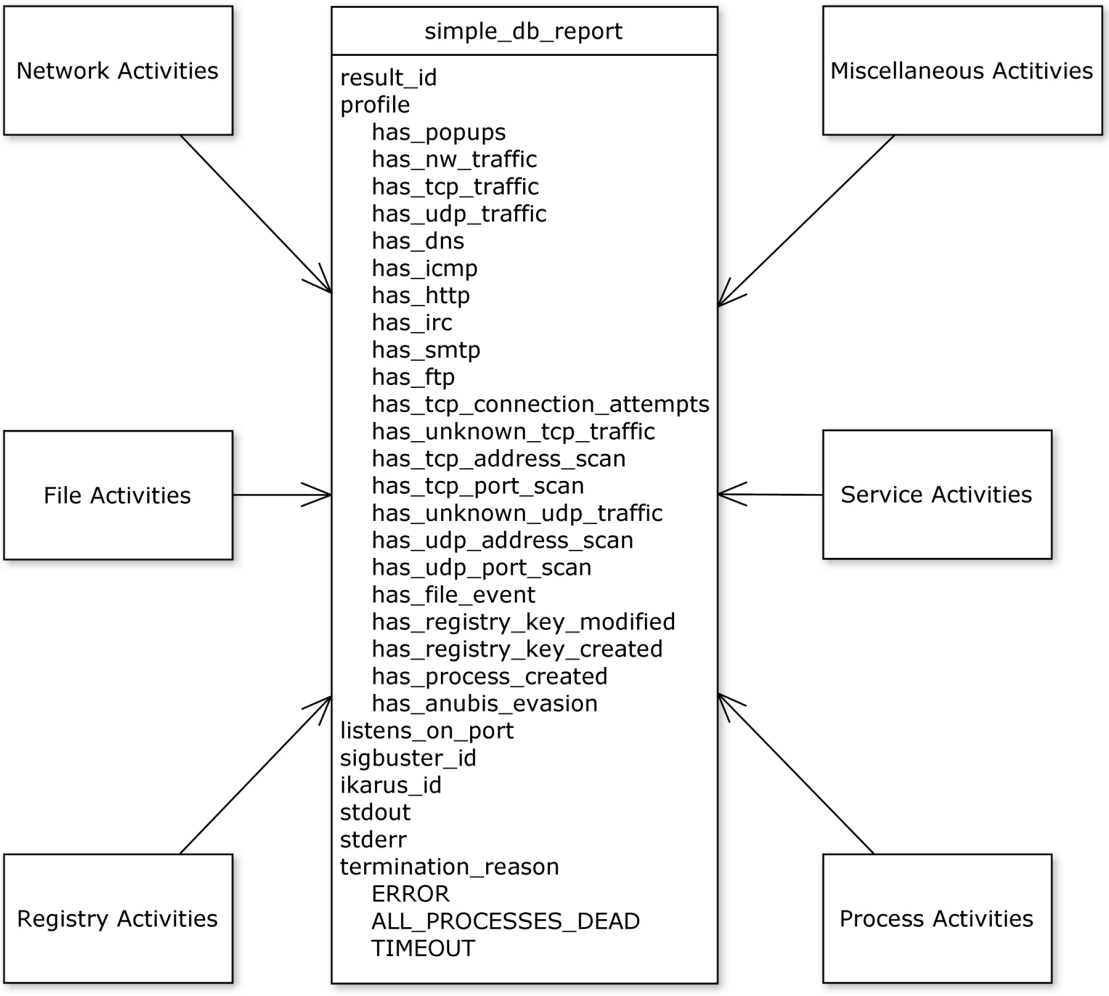


Figure 3.1: Overview on the Database Schema

As illustrated in Figure 3.1, the schema consists of a central table that holds general data like the sample's output and summary information on the analysis that is not maintained in the database already. This summary data includes the label that the antivirus software by Ikarus assigns to the executable, the listening port, if one is opened to listen for incoming network connections, and boolean values if a sample has network activity or registry activity. This summary data is required by our search system and allows quick categorization of the samples without looking into more detailed data.

The parameters of each type of system call are stored in a separate table that references to `simple_db_report.result_id`. In order to save storage space we exploit the fact that some parameters of system calls are not only shared between one type of system call, but they are often shared amongst different types of system calls. A complete description of the database schema is described in Appendix A but Figure 3.2 shows an extract of the database schema that illustrates how a file name is shared amongst system calls that create, delete, modify or read a file. We apply this technique not only to file names but use it for service names, registry keys names, registry values, registry value names, mutex names, and process names as well. This technique would not save storage space for parameters that are very diverse, but we observe that the identified parameters are reused to a major part and that this technique thus tremendously decreases the overall storage space required for the behavioral data. Furthermore, this technique gains in efficiency if the number of stored reports increases. For each additional report that is added to the database, it more likely that a file has been accessed by a previously analyzed sample or a service with an identical name has been created previously.

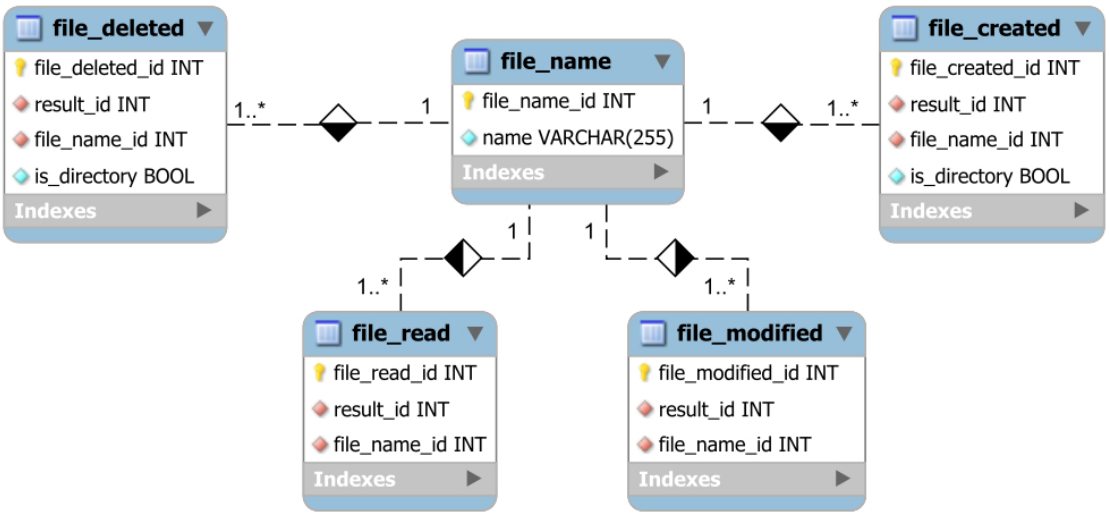


Figure 3.2: Parameter sharing

By reducing the complexity of the analysis report as we parse it into the database and by concentrating only on information that is not maintained in the database already and by exploiting properties of the analysis report, we manage to reduce the storage space required by the behavioral data of 3 million analysis reports to 14 GB. The compressed XML reports require 4 GB more

storage space even though the database is not compressed. Additionally, this storage engine favors efficient access to the stored data.

3.3 Indexes

Indexes are required in relational databases to efficiently retrieve entries of large tables [64]. Abstaining from indexes would require MySQL to look at each entry sequentially in order to retrieve the entries in question [16]. When running queries against tables with more than 20 million rows, as in our case, indexes are an absolute necessity in order for queries to perform well.

We use three different types of indexes that MySQL supports on MyISAM tables. A **primary key index** is used for primary keys, an **unique index** ensures that no duplicate entries can be stored in a table and a regular **index** allows duplicates.

We utilize MyISAM as storage engine of the MySQL database. MyISAM does not support foreign key constraints [64]. However, our queries frequently require the joining of multiple tables. In order to boost the performance of table joins, we have to maintain indexes on the fields that reference entries in foreign tables.

Additionally, we add unique indexes to those fields that hold values that we want to store once only in order to avoid redundancies and reduce the overall storage space. These fields are storing file names, registry keys and values, and service names for example and are the most interesting parameters of the monitored system call and are thus the values that are used as selectors most often. These indexes do not only ensure that identical values are stored once only, but increase the performance of queries that are run frequently as well.

Other common selectors are IP addresses. Frequently, we query for samples that contact certain IP addresses or ranges of IP addresses. The description of the tables that is given in Appendix A reveals that IP addresses are stored in a number of tables - `smtp_conversation`, `tcp_connection_attempt`, `http_get_data` and `irc_conversation` to name only a few. We cannot use an unique index in this case, as it is legitimate to contact a single IP address multiple times and thus save multiple entries with the same IP address in the database. Instead, we use a regular index for fields that store IP addresses.

In total, all these indexes require 22 GB on disk. Although, we require more storage space for indexes than we require storing the actual data, the performance that is gained by utilizing these indexes pays off.

3.4 Data Cleansing and Transformation

In the analysis report, all the behavioral data is saved as string values due to the nature of XML. Some of these values represent integers and others represent dates. In contrast to XML, a relational database, like MySQL, offers multiple data types. In order to store the data efficiently in the database, we have to identify the type of the values and transform the values accordingly.

The identification and transformation of integer values is obvious and an easy task. But we have to consider the maximum number that we expect for a certain value. For example, we log the number of certain exceptions that occurred, the number of a key that was checked and

the number of read accesses to a certain file or registry value. These numbers are likely not to exceed 65 535, the maximum value the data type `unsigned smallint`, but since we use the C++ data type `integer` to count the occurrences, the numbers might exceed this value. But the number of occurrences cannot drop below zero. Therefore, we can use the data type `unsigned int` to store these values. Port numbers of the IP protocol on the other hand will by definition never exceed the maximum value that can be represented by the data type `unsigned smallint`. In the XML report, we store IPv4 addresses in their dotted decimal representation. This representation makes it very difficult to query for ranges of IP addresses. Therefore, we convert the dotted decimal representation of IPv4 addresses to their integer representation and store them as `unsigned int`.

Some string values in the XML report represent predefined values. If a sample opens a port, we know that a valid protocol for the port can either be 'UDP' or 'TCP'. We can exploit this knowledge and define an enumeration of valid values for this column in the database. This strategy saves storage space as we have to save the possible values only once and can reference the value with its index for each entry. Additionally, this strategy makes queries on these entries more efficient. Therefore, we apply this strategy on all values where it is appropriate. For some values, like the MIME type of email attachments, we have to split the values in order to apply this technique. IANA lists eight possible content types that we can save as an enumeration [18]. The list of possible subtypes of the MIME type on the other hand, cannot be maintained efficiently and this part of the MIME type is thus stored separately as a string value.

Information on HTTP connection is already maintained in the database for the system FIRE [14]. This system needs data on HTTP connections in more detail than it is available in the analysis report. Therefore, the information is extracted directly from the network dump produces by the analyzed samples. Because this data is available in the database anyhow, we do not have to extract the less detailed information on HTTP connections from the analysis report. However, in the course of this work we optimized these tables as well. Previously, all data was saved as a string data type in the database. We applied the same techniques to this table as we utilized for the rest of the database. Thus, we converted the IPv4 address to the data type `int`, represented the port number as `smallint` and parsed and adapted the time zone of the server date.

To import the data into the database we implemented a Python script that parses the analysis report after the analysis of a sample is complete and that converts the data accordingly. Another script parses the network dump and extracts the HTTP conversations and inserts these into the database. We adapted and optimized this script in order to cope with the changes made to these tables in the database. In order to insert 2 million reports that have been analyzed prior to this work, we created a script that runs the script that parses the XML report for each analysis report. Because Anubis is subject to continuous improvements and development, the structure of the analysis report has been affected as well. We reflect these changes by versioning the analysis report. Currently, Anubis produces reports of version 3.1. In order to maintain backwards compatibility to some extent, we support the parsing of reports since version 2.0, which has been introduced in September 2007.

3.5 Validation and Testing

In order to verify the correctness of the script that imports the data of the analysis report into the database we used eight analysis reports listed in Table 3.1. This set of reports covers all actions that are logged in the analysis report and should be inserted into the database. We manually checked whether all actions are being inserted correctly.

	apitest	**Backdoor.Hupigon.ALU**	**Backdoor.Rbot**	**Trojan-GameThief.Win32.Magania**	**Virus.Win32.Small**	**Trojan-Dropper.Win32.Microjoin**	**Backdoor.Win32.Iroffer**	**Trojan-Dropper.Logsnif.A**
address_scan						X		
device_control_communication	X	X	X	X	X	X	X	X
directory_created	X						X	
directory_monitored					X			
directory_removed	X							
dns_queries			X		X	X	X	X
driver_loaded				X		X		
driver_unloaded				X				
exception_occurred		X	X	X				
file_created	X	X		X	X	X	X	X
file_deleted	X		X			X		X
file_modified	X		X	X	X	X	X	X
file_read	X		X	X	X	X	X	X
file_renamed				X			X	
foreign_mem_area_read	X	X	X		X	X	X	
foreign_mem_area_write	X	X	X		X	X	X	
fs_control_communication			X	X		X	X	
ftp_traffic								X
http_traffic		X			X	X		X
icmp_traffic								X
irc_traffic			X					
key_was_checked							X	
link_created							X	

	apitest	Backdoor.Hupigon.ALU	Backdoor.Rbot	Trojan-GameThief.Win32.Magania	Virus.Win32.Small	Trojan-Dropper.Win32.Microjoin	Backdoor.Win32.Iroffer	Trojan-Dropper.Logsnif.A
mutex_created		X	X	X			X	
opened_port						X	X	
popup_window			X	X		X	X	
port_scan								X
process_created	X	X	X		X	X	X	
process_killed	X							
reg_key_created	X			X	X	X	X	
reg_key_created_or_opened	X					X		
reg_key_deleted	X							
reg_key_monitored			X		X	X	X	X
reg_value_deleted	X							
reg_value_modified	X		X	X	X	X	X	
reg_value_read	X	X	X	X	X	X	X	X
section_object_created	X	X	X	X	X	X	X	X
service_changed	X					X		
service_control_code	X			X		X		
service_created	X			X		X	X	
service_deleted	X							
service_started	X			X		X		
smtp_traffic		X						
tcp_connection_attempt		X	X		X	X		X
unknown_tcp_traffic		X	X		X	X		X
unknown_udp_traffic		X	X	X	X	X	X	X

Table 3.1: Test reports

In the course of testing we found a minor bug in the presentation of the analysis report. In the XSL style sheet that we utilize to display the analysis report, we had a typing error that prevented foreign memory area writes to be listed in the report. Additionally we fixed some typing errors in the database schema and the script that parses the XML report into the database until the set of test reports was inserted into the database successfully.

CHAPTER 4

Statistics Representation - A Didactical Approach

The problem sets and relationships we want to envision often reside in the three spatial dimensions of our world, but paper or computer screens are flat and hold room for two dimensions only. Multiple dimensions have to be mapped to a two-dimensional layout to fit these communication channels. The finite space available on paper or screen calls for an increased data density [68]. Nevertheless, visualizations are a powerful form of communication. [31] describes an experiment where physics problems had to be solved with and without supporting visualizations and diagrams. The study shows that the diagrams aided the test person in grouping information. In addition, extensive scanning for symbolic labels could be avoided. The result is clear: visualizations reduce the time necessary for understanding of often abstract facts and thus support the learning process. The advantages of visualizations can be summarized as follows:

- Visualizations can depict huge amounts of data. It is simple for the viewer to extract the important information instantly.
- Visualization favors the discovery of previously unforeseen properties of data.
- Visualization allows detection of erroneous data. The deviations are revealed and if they occur unexpectedly, they suggest a faulty database or measuring mistakes. This advantage makes visualizations very important for quality control.
- Visualization aids understanding of features of the data and allows detection of patterns in this data.
- Visualization allows formation of hypotheses.

To produce visualizations that make use of these advantages, properties of visual perception have to be taken into account. Special guidelines of information visualization have to be followed

and special care has to be taken when mapping data tables to visual structures. The visualization should be expressive and represent the data only, not more and not less, thus being unambiguous [31]. This is not an easy task, because it is easy to introduce undesired data or relationships when choosing an unexpressive representation. One of the most common mistakes occurs in mixing nominal, ordinal, interval and ratio scales, which proposes unintended relationships. By no means do we intend to give a complete overview on the subject of information visualization and the human visual perception, but we find it necessary to highlight important basics that are the precondition for sound visualizations.

4.1 Information Visualization

Stuart defines information visualization as the 'use of interactive visual representation of abstract, nonphysically based data to amplify cognition' [31]. Initially, we have to emphasize properties of visual representations that allow suitable representation and how these properties are perceived by the human visual system.

The human visual perception is powerful. While changes are detected very fast, it takes much longer to perceive rigid information. An evolutionary explanation for this phenomenon might be that it was more important for our ancestors to promptly react to changes in environment, like approaching animals, than it is to constantly analyze a still and thus potentially less dangerous environment. It seems reasonable for the brain to concentrate only on important and new information and optimizing visual perception for this task rather than wasting resources on old and already familiar information. The human visual system is very efficient in detecting and distinguishing patterns as well, if they are presented correctly. Detecting and distinguishing patterns plays a major role in information visualization. Therefore, we want to shed light on the elements that are characteristic of a visualization that can be understood and processed quickly and efficiently.

Preattentive Processing

Preattentive or automatic processing happens before conscious attention and serves pattern recognition, perceptual organization and object identification. This process runs automatically and objects can be already visually identified after a very brief exposure. The fact that something is said to be preattentively processed if the response time of finding a target surrounded by distractions is very low, makes preattentive processing very attractive for visualization. If the target information is highlighted in such a way that it can be processed preattentively, it can be extracted very efficiently, usually in less than 10 milliseconds per item [70].

Many experiments have been conducted to find features that are processed preattentively. Ware lists four categories for such features: form, color, motion and spatial position [70]. The category form embraces most features: line orientation, line length, line width, line co linearity, size, curvature, spatial grouping, blur, added marks and numerosity. Color consists of hue and intensity, motion can be achieved by flicker or direction of motion and spatial position includes 2D position, stereoscopic depth and convex or concave shape due to shade. Of course, it is desirable to encode data in such a way that it can be processed automatically. Table 4.1 gives an

98103901236454062348069124 9	98103901236454062348069124 9
11 9 57087417961382088044931 8	11 9 57087417961382088044931 8
80234829688485149214828171 6	80234829688485149214828171 6
29007414774712348 9 5637 3 5128	29007414774712348 9 5637 3 5128

Table 4.1: Preattentive processing (left) versus attentive processing (right)

(a) Integral dimensions color and shape

(b) Separable dimensions location and color

Figure 4.1: Examples for integral and separable dimensions. [70]

example of how counting the fives in the left column of the table can be supported by preattentive processing. While it is necessary to sequentially scan each digit in the right column of the table to identify all the fives, the same can be achieved on the left side merely by scanning for red digits.

But not all features are equally viable for representation of data. While spatial grouping is the most effective type in general [31], the type of data influences the most suitable property decisively. It would be very ineffective to encode data of temperatures given in Celsius as different shapes. This data resides in an interval scale and could be better expressed by size or color. Encoding data of a nominal scale as different shapes works perfectly well. Traffic signs or pictograms would be an example for such encoding.

It is possible to highlight different information sets by using different preattentively processed features. One has to beware though, not to overload a visualization. Using too many distinct preattentively processed features may result in bad distinction between the separate information sets. Additionally, one has to keep in mind that some features work well with each other, while other combinations cannot be processed automatically anymore. Mixing shape and color of objects as in Figure 4.1a is an example for the latter and they form integral dimensions. It is not easy to distinguish the red circles from the black circles. Figure 4.1b on the other hand uses spatial grouping and color of objects. These properties can be well distinguished and they form separable dimensions [49].

Attentive Processing

Attentive or controlled processing requires consciousness. In contrast to preattentive processing, this process is slow and has a very limited bandwidth and is prone to inhibitions [31]. The observer deliberately seeks to extract relevant information from the visualization. He draws

conclusions and connects the visual information with verbal information. Each element of a visualization is analyzed sequentially.

Although preattentive processing is often preferred, as it enables to quickly transport the relevant information to the viewer, attentive processing can be necessary and desired in some cases, like decision making. Tufte gives an example of a person choosing a t-shirt of a set of different children's t-shirts. Multiple reductions of the representation of these t-shirts of different color give a complete overview of the available models and aid the decision process, as single representations - possibly distributed among multiple pages - do not have to be memorized, but the distinct features can be compared directly [68].

4.2 Visualizing Time-Dependent Data

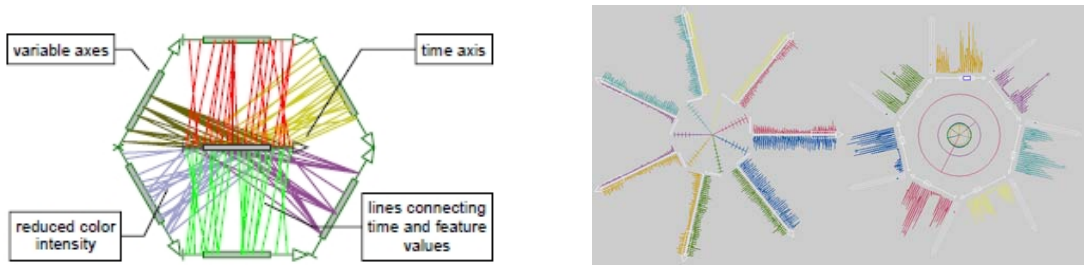
The aim of visualizations of temporal data is to answer properties of data elements like occurrence in a specified time frame, existence or time span of their occurrence, their repetitions, grouping with other data elements, temporal dependencies in the order of their occurrence and the rate at which the data elements change [27]. We want to answer these questions for malware behavior.

It would be natural to encode time-dependent data as animation, as that is what happens in nature. The animation can be in real-time, it can be a time-lapse visualization where time is condensed or it can be a slow-motion visualization. Time-lapse visualizations make sense when a process is depicted that takes very long and no sudden changes of data are expected. On the other hand, slow motion visualizations come handy when a process is depicted that happens very fast and involves rapid changes of data values. However, encoding time-dependent data as an animation is often not very effective, because it does not allow direct comparison of two data elements [31]. Mapping the time to space instead is usually more effective.

Mapping time to space is especially more effective for the visualization of malware behavior. Currently, the data occurs in a time-span of three years. The only feasible method to animate trends of malware behavior is as a time-lapse visualization. However, we cannot rule out that sudden changes in malware behavior do take place.

In a static representation of time-dependent data, the change of the data can be represented as curves or graphs and thus allows making better conclusions about the curse of the underlying data. However, the representation is highly contingent on the properties of the time axis. In [37], A.U. Frank distinguishes several types of time. Time can be described as discrete time points or as time intervals like an hour, a day or a month. Furthermore, time can be defined by a starting point and an end point in contrast to cyclic properties as with the seasons of a year. With ordinal time, we can only conclude whether an event occurred before or after another event. Continuous time on the other hand allows quantifying the difference between these two events.

Although an analysis of a malware sample completes in a discrete point in time, mapping the behavior of each sample separately to the visualization would not be very meaningful. Nevertheless, visualizing large data sets will likely result in overcrowded and cluttered visualizations [28]. Instead, we want to aggregate the data and quantify the occurrence of a specific behavior in a time interval. In particular we are interested in daily and monthly intervals. Additionally, the period under observation has a defined starting point and an end point and thus uses linear time. Considering the fact that the observed time intervals can be linked to a specific date, we can



(a) TimeWheel visualization (b) MultiComb visualization

Figure 4.2: Example visualizations for time-dependent data [51]

quantify the time difference between two intervals and can conclude that we have to deal with continuous time.

We have found that we deal with linear and continuous interval time. Several visualization methods can be applied to depict these time properties [51]. In particular, TimeWheel and MultComb visualization techniques look promising. A TimeWheel visualization is shown in Figure 4.2a. The center axis represents the time while the surrounding axes represent one variable each. The MultiComb visualization technique is depicted in Figure 4.2b and uses a separate time axis for each variable and aligns these in a circular way. However, we plan to observe at least 30 different malware behaviors which would result in 30 different variables. This magnitude of variables would make these representations very complex. We could overcome this obstacle by presenting six to eight different behaviors at a time only and give the analyst the opportunity to change the variables that are currently shown. However, we want to present these variables in different scales. Since the absolute number of observations of a certain behavior is highly dependent on the total number of analyses available for this time interval, we want to offer a percentage value as well. Unfortunately, multiple scales are not supported very well by these visualization techniques.

Alternatively to those techniques, we could use Parallel Coordinates. In this technique, one axis is used for each variable and all axes are arranged one upon the other. Although it would be possible to represent variables in different scales witch this visualization technique, the visualization of more than 30 variables would get very large - too large to fit on a computer screen.

Therefore, we decided to group relative variables into separate line graphs with a time axis and two y-axes to represent variables in percentage values and absolute values. By opening multiple line graphs simultaneously, the analyst can simulate Parallel Coordinates with few variables. Additionally, line graphs exploit several separable dimensions and thus allow the viewer to get an idea of the underlying data preattentively. In line graphs we can make use of the preattentive features color, spacial position, line orientation, curvature, and added marks [70].

4.3 Visualizing Proportional Data

The aim of visualizing proportional data is to present a quick overview on its distribution. Tables, bar charts and pie charts are capable of transporting this information [32]. In tabular form, one column lists the values and the value's labels are listed in a second column. In a bar chart, the height of a bar indicates the magnitude of the value it represents. A pie chart represents the values as slices in a circular form. The size of this two-dimensional area represents the proportion of the value it represents. While tables and bar charts allow efficient comparison of values, comparing values in a pie chart is not supported very well, since it is difficult to compare the size of a two-dimensional area or the angles of the slices [35]. However, sorting the slices by size counteracts this obstacle. Nevertheless, pie charts are used seldom in scientific publications and E. Tufte recommends not to use pie charts at all [69].

We have an additional demand on the visualization of proportional data. We want to present an overview of the distinctness of certain properties of malware behavior, like the modification of system files, by ranking their top ten. However, a frequent task will be to identify the number of distinct values for a property that sum up to 25 percent, 50 percent or 75 percent. By relying solely on tables, this task would involve additional calculations. Using bar charts, the viewer had to stack up the bars in his mind, which is by no means more efficient that adding up the values. Pie charts, on the other hand, offer support for this task, provided that the slices are in the correct order. Here, it is easy to identify the number of slices that are required to fill a half or a third of the pie. For this reason, we decided to visualize the top ten statistics as pie charts and exploiting the pie chart's legend as a table by displaying the percentage values as well.

4.4 Concepts of Interactivity

In contrast to paper, where visualizations remain static, a computer screen offers multiple opportunities for dynamic and interactive display of visualizations. C. Ware distinguishes among two categories of interaction techniques. The category "data selection & manipulation" contains interaction techniques that support the discovery of data. On the other hand, the category "exploration & navigation" deals with interaction techniques that modify the view on the data and aids the better understanding of the visualization [70].

Although the main research area of interaction techniques regards the interaction with 3-dimensional models, several interaction techniques can be applied to visualizations of time-dependent data. In particular, we are interested in interaction techniques that improve visualizations of line graphs.

A line graph usually represents a large number of values. If the range where data occurs is large it is difficult to find the exact value of the underlying data. Attaching a written value to each data point would result in an overcrowded and unreadable visualization. Instead, we can use hover queries to display the value of the data where the mouse points to.

Exploring the data and navigating in time clearly is the main purpose of time-dependent line graphs. A number of interaction techniques can aid this task:

- **Viewpoint Control**: This interaction technique allows viewing the data from different

angles. In the case of time-dependent data, it can be used to navigate forward and backwards in time.

- **Distortion Techniques**: One of the most important interaction possibilities in time-dependent visualizations is to offer the data in different time granularities and thus provide an intelligent zooming [28].

- **Rapid Zooming Techniques**: A line graph can represent a very long period of time. However, the focus lies on different, possibly smaller, time periods of time during different stages of the exploration. This interaction technique allows for zooming into the dataset and representing only a subset. To us, this is the most important technique, along with the viewpoint control.

- **Elision Techniques**: This technique describes the representation of a large object that is collapsed into a smaller object. A line graph may hold multiple lines. The user might not be interested in all lines equally at the same time. This technique allows the user to hide single lines and show them again if desired.

- **Brushing Techniques**: This technique allows highlighting of a graphical structure. In a line graph with multiple lines a single line can be distinguished better from the other by highlighting it than simply by different colors.

The visualization of proportional data is less complex and does not offer many possibilities for interaction techniques. This is especially true for static data. With dynamic data, we could provide the user with the opportunity to change the interval as necessary, but with fixed intervals, this is not possible. Hover queries, on the other hand, can be used to enrich visualizations of proportional data as well.

4.5 Demands on Charts

During the research, we found that we require two types of graphs to represent malicious behavior.

We need line charts that are capable of representing multiple graphs in two scales, absolute numbers and percentage values. The line chart should be adaptable to present a stacked line graph as well. In addition to these basic requirements on line charts, we want to be able to view only a subset of a graph's values at a time, allowing investigation of a smaller time-span in more detail. Viewing a smaller time-span we want to be able to move this time window forward and backwards in time. Because we want to present multiple graphs in a single chart, giving the viewer the possibility to hide and show single graphs on demand in order not to get distracted by other graphs in possibly different scales is our final requirement on line charts.

For a pie chart, we require the legend to be presented in tabular form. This allows the viewer to get a quick overview on the distribution and the ranking of the displayed data.

4.6 Tool Evaluation

After specifying the demands on charts, we searched the web for charting tools that would meet our requirements. In Table 4.2, we list the products we found along with the requirements they fulfill.

Product	**Line chart**	**Stacked line chart**	**Zoom**	**Scroll**	**Elision**	**Pie chart**	**Tabular legend**
amCharts `http://www.amcharts.com`	X	X	X	X	X	X	X
Animated Chart `http://www.animatedchart.com`	X					X	
AnyChart `http://www.anychart.com`	X	X				X	
Flash Charts Pro `http://www.web-site-scripts.com/flash-charts`	X	X				X	X
Fly Charts `http://flycharts.net`	X		X	X	X	X	
Fusion Charts `http://www.fusioncharts.com`	X		X			X	
Open Flash Chart `http://teethgrinder.co.uk/open-flash-chart`	X					X	
Swiffer Chart `http://swifferchart.yaaman.com`	X	X				X	X
XML/SWF Charts `http://www.maani.us/xml_charts`	X	X	X	X	X	X	X
zxChart `http://www.ankord.com/zxchart.php`	X					X	X

Table 4.2: Evaluation of charting tools

We found only two charting tools that meet the requirements entirely, *amCharts* and *XML/SWF Charts*. For a final decision, we tested both tools extensively. Although they are almost equally powerful, we chose amCharts for two reasons: First, amCharts supports CSV and XML formats for the input data while XML/SWF Charts lacks the support for a CSV input format. However, the CSV format is handy, as it can be used as input for other statistics applications as well and does not require prior conversion. Second, it is not possible to test the interaction

features of XML/SWF Charts in its trial version, because a click anywhere on the chart leads to the product's home page. amCharts, on the other hand, only displays a link to its home page above the chart in its trial version.

CHAPTER 5

Design and Implementation of an Extensible Framework for Creating High Level Views of Malicious Behavior

In this chapter, we will start off by describing the general architecture of Anubis and we will then point out the components that are used by the implementation of a framework for creating high level views of malicious behavior. We will describe its design with special focus on its extensibility.

The architecture of Anubis consists of the following components:

- The **analysis server** acts as an interface to the outside world. It receives the analysis tasks and coordinates their analysis. An analysis task can be a Windows executable or an URL. The submissions can be made via a web interface. Additionally, Anubis accepts automated submissions of executables via the submit handler of nepenthes [19]. The analysis server manages the analysis workers and triggers the start of the analysis of a submitted task whenever a worker is idle. As soon as the analysis is complete, the analysis server returns a detailed report, describing the actions performed by this submitted binary.

- Anubis runs several **analysis workers**. The workers are responsible for the analysis of a sample. For each analysis, a worker starts a virtual machine running Windows XP. All security relevant Windows native system calls and Windows API functions called by the sample are recorded. In the case of an URL analysis, Internet Explorer opening this URL will be analyzed. Whenever the sample under analysis terminates or when the analysis process reaches a timeout, the worker creates the analysis report and saves a dump of the generated network traffic and waits until the analysis server assigns it the analysis of the next sample.

- All network traffic that is generated during the analysisis categorizes as harmless or rogue network traffic. All rogue network traffic will be redirected to the **victim server**. It runs nepenthes and is configured to accept all incoming connection attemts, while it will not forward this traffic to their intended recipients. This way it is possible to simulate vulnerable hosts and SMTP servers without doing any actual damage or forwarding any emails.

- A **storage server** stores the submitted samples, the analysis reports and the network dumps.

- The **database server** is used by the analysis server to coordinate the analysis tasks and holds information on the submitted samples.

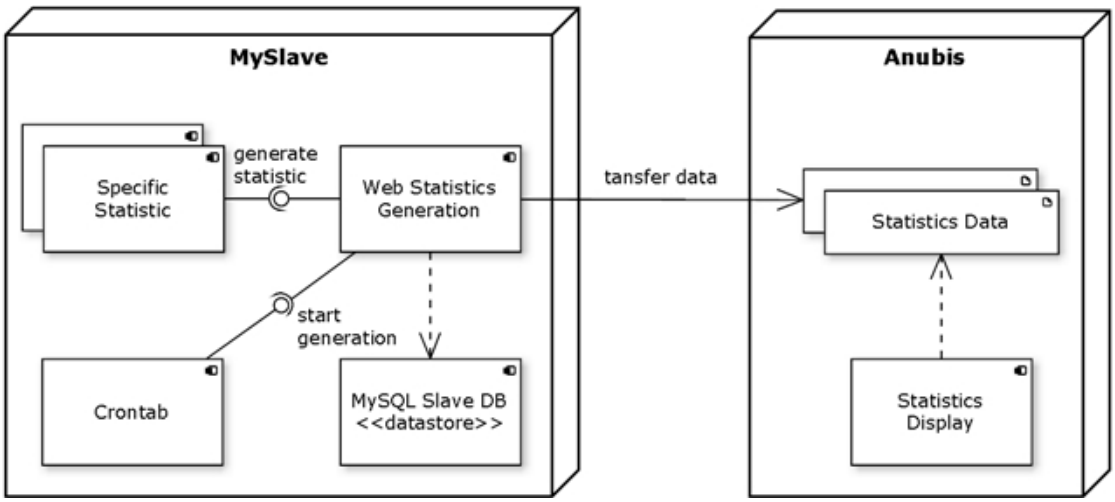


Figure 5.1: Architecture of the statistics framework

The architecture of the framework is distributed among only two servers. As illustrated in Figure 5.1, the generation of the statistics happens at 'MySlave'. MySlave is the slave database server of Anubis. We use the slave server for the calculations of the statistics to reduce the load on the primary database server.

The statistics are part of the Anubis management interface. Therefore, it is hosted along with the management interface at the Anubis analysis server.

In the following sections, we will describe the implementation of the framework in more detail and explain the necessary steps at each location of the statistics framework to add additional statistics. Generally, a single statistic comprises the following components:

- **`statistic_name.py`** that is located in the directory `~/queries/` in the home directory of `anubis-stats` on MySlave.

- **`statistic_name_mode.csv`** files hold the data that is displayed in the chart. They are stored in the directory `~/data/` for line charts or `~/data/statistic_name/` for pie

charts in the home directory of `anubis-stats` on MySlave and they are transferred to the directory `stats/data/` located in the trunk folder of Anubis' management interface on Anubis.

- **`statistic_name_settings.xml`** stores the settings for the statistic's chart like the labels of the graphs and their scales for line charts. Pie charts do not require a settings file. It is located in the directory `stats/settings/` in the trunk folder of Anubis' management interface on Anubis.

- **`statistic_name` variable** in `stats/incs/config.js` in the trunk folder of Anubis' management interface on Anubis. In the head of this JavaScript file, the variable specifies the affiliation to the statistics category and the position in the menu. Additionally, the variable holds information about the type of statistic, its title, the available modes and its summary text.

5.1 Query Environment

The query environment is located in the home directory of the user `anubis-stats` on MySlave. The main entry point for the query environment is `web_statistics.py`. It is executed as a cronjob once a day. The statistics always represent the data up to the previous day. To keep the statistics as current as possible, we run the cronjob shortly after midnight. But the time and date values in the database are given in Coordinated Universal Time (UTC). Unfortunately, it is not possible to schedule a cronjob with UTC, instead it relies on the server time of the machine it runs on, which is UTC+1 or UTC+2 during daylight saving time. This is the reason why execution of `web_statistics.py` is scheduled daily for 02:15.

The script `web_statistics.py` initializes `query_helper.py`, which manages the database connection and provides functionality that is required by many statistic scripts. For most of the statistics we assume that the projection of the data in the database is complete for previous days. New analyses always get the current time as time stamps, and if an analysis is complete then it will not be modified at a later point. To assert this prerequisite, `query_helper.py` first checks whether an analysis has completed for the present day. If there is at least one analysis that completed on the current day, we can safely assume that the past days are complete. If `query_helper.py` does not find a completed analysis for the current day, the database might lag behind and thus cannot be sure that previous days are represented completely in the database. In that case an error is reported and the generation process is aborted.

The second task, apart from setting up the database connection, which is completed during initialization of `query_helper.py` is the update of a set of CSV files that cache the number of reports that are saved in the database for a certain time interval. These numbers are frequently used by the statistic generation scripts to calculate percentage values.

In the next step, `web_statistics.py` calls the function `generate_statistics` of every Python script that is located in the folder `~/queries/`. A reference to the previously initialized `query_helper.py` is passed to these scripts as an argument.

To add another statistic to the framework an additional Python script has to be placed into the directory `~/queries/`. The function `generate_statistics` has to be implemented

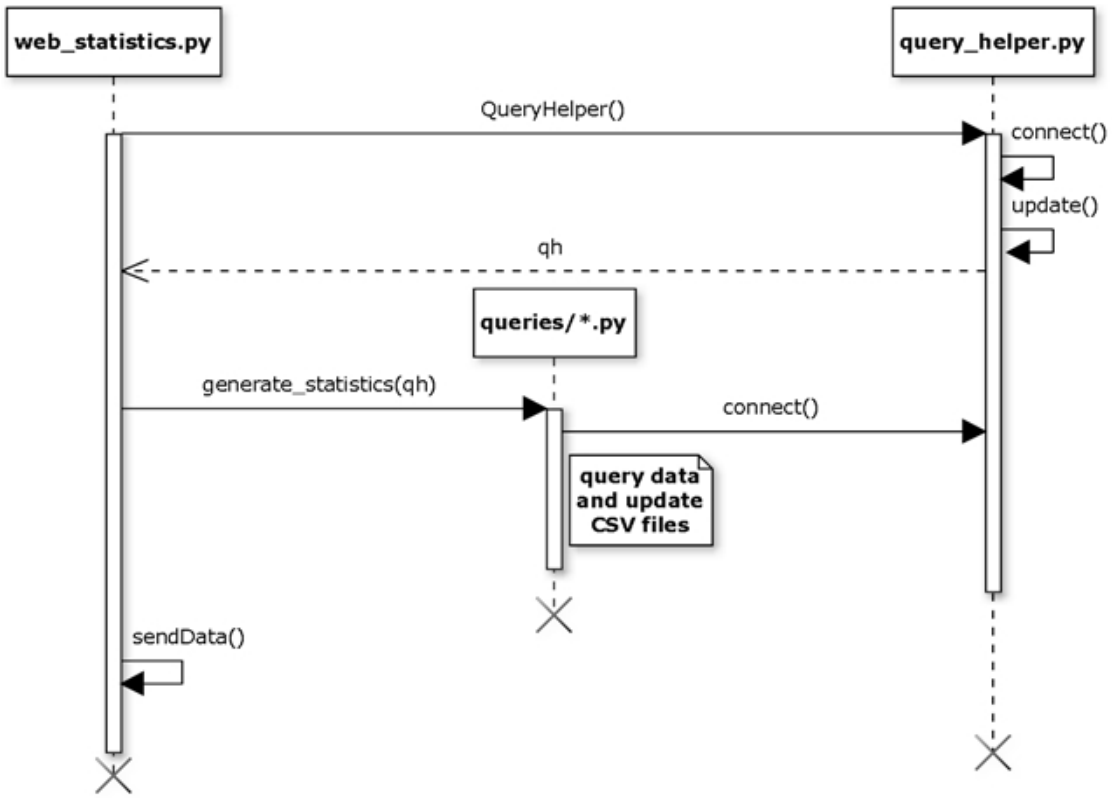


Figure 5.2: Sequence diagram of the statistics generation process.

and must invoke the generation of the CSV files and thus the query to fetch the required data from the database. Additionally, all statistics generation scripts provide a `main` function that initializes `query_helper.py` and calls the `generate_statistics` function. This allows any specific statistic generation script to be run manually and separately.

The following functions of `query_helper.py` are available to the statistic generation scripts:

- **`connect`** requires no arguments and initializes a database connection and returns it or simply returns the database connection if it has been initialized already. Using this function, the statistic generation scripts can share a single database connection and do not have to each set up a connection separately.

- **`start`** requires the name of the statistic that is currently executing as a parameter. This function saves the start time of the statistic execution, which is later required to compute the complete execution time of the generation of a single statistic for the log file.

- **`log`** requires the name of the statistic and the current mode of the statistic that has been completed and writes the time that the execution of this mode took to the log file. Usually,

a statistic is available in several modes - daily, monthly and a complete mode. Each of these modes presents the data in different intervals.

- **`end`** requires the name of the statistic that is currently executing as a parameter and writes the complete runtime of the statistic to the log file.

- **`getLastDate`** requires the name of a CSV file as parameter and returns the day or the month for the latest values available in this file.

- **`getMonthlySum`** requires the path to the CSV file that holds the data in daily mode, the month from which on the sums are required and a list of columns of the CSV file that should be summed up. It returns the following data structure: `[[['month', #reports for this month], {x: sum of column x, y: sum of column y,..}]]` where month is of the form `mmm-yyyy`.

- **`getTotalSum`** requires the path to the CSV file that holds the data in monthly mode and a list of columns of the CSV file that should be summed up. It returns the following data structure: `[#total reports, {x: sum of column x, y: sum of column y,..}]`.

The process of statistics generation is different for pie charts and line charts.

Usually, a line chart utilizes CSV files for three different modes - daily, monthly, and total. The first thing for the statistics generation script of a line chart to do is to get the latest date that is saved in the file `statistic_name_daily.csv`. It will query the database only for samples that were analyzed at a later point in time. If this file does not exist, it will be created and the database will be queried for all samples. The monthly and total values will be calculated using the functions `getMonthlySum` and `getTotalSum`. Percentage values can be calculated using the number of reports per month or the total number of reports. In the most cases, as with the statistics on system file modifications or HTTP activity, the monthly and total values do not require additional database queries except when the statistic does not display absolute values but percentage values only. Additional database queries are necessary in this case.

A pie chart utilizes a number of CSV files. All the CSV files for a pie chart are stored in a separate subfolder with the same name as the name of the statistic. The file `statistic_name.csv` holds a comma separated list containing each month for which a pie chart is available. A separate CSV file for each of those months is required for a pie chart. The statistics generation script of a pie chart queries the database for all months in which samples have been analyzed. It checks for each month whether an according `statistic_name_mmm-yyyy.csv` file exists. If this file does not exist, it will be created. Additionally, the CSV file of the current month will be recreated to keep it up to date. Two database queries are required to create the CSV file of a month. First, the generation script queries the total number of samples that have a certain characteristic and second, the top ten alterations of that characteristic are selected together with the number of their occurrence. Finally, percentage values are calculated and the results are written to the CSV file.

Finally, after completion of the update of all statistics, `web_statistics.py` connects to Anubis propagates the updated CSV files to the web server.

5.2 Charts Environment

In order to view the charts environment, a user requires a current browser with JavaScript enabled and an Adobe Flash Player plug-in of version 8.0 or higher. We use amCharts, a Flash applications, to display the charts [4].

The charts environment is located in the folder `stats/` in the trunk folder of Anubis' management interface on Anubis. It comprises the following components:

- **`stats.html`**: The charts environment is displayed in this HTML file.
- **`incs/stats.js`**: This JavaScript file provides functionality for the construction of the menu and for the change between statistics and modes of statistics. Additionally, it processes the communication between the charts environment and the Flash files.
- **`incs/config.js`**: This JavaScript file holds information for each statistic. It knows whether a statistic is a line chart or a pie chart and the available modes for a statistic, returns the name and a short description of a statistic and assembles the summary text of a statistic.
- **`amChart/amline.swf`**: This Flash object is responsible for the display of line and area charts.
- **`amChart/ampie.swf`**: This Flash object is responsible for the display of pie and donut charts.
- **`amChart/swfobject.js`**: This JavaScript file is used to load the Flash charts and displays an error message if an older version of Flash is installed.
- **`data/`**: The CSV files that have been generated on MySlave by the query environment are saved in this folder.
- **`settings/`**: The setting files for the charts are stored in this folder as XML files. `settings.xml` holds general settings for line charts. Settings in this file can be extended or overridden by `statistic_name_settings.xml`, which contains the labels of the graphs and their scale at least. An individual setting file for each pie chart is not necessary, instead, `psettings.xml` holds the settings for pie charts. A complete reference for the settings of line charts is provided at [5] and a detailed description of the settings of a pie chart is available at [6].

When `stats.html` is loaded, the menu, which contains the links to all statistics available, is built according to the `statistics` array in `incs/config.js`. This array specifies the order of the statistic and their affiliation to categories. The display name and the short description of a statistic are retrieved from this JavaScript file as well. An exemplary configuration of a statistic is provided in Listing 5.1. The difference to a configuration of a pie chart is that it has to return `"pie"` instead of `"line"` as type and the functions `getTotalText` and `getModes` are not required.

Whenever a statistic is clicked, the script `incs/stats.js` first checks whether the statistic is of the same kind as the statistic that is currently displayed. If the types differ, the according

Flash object has to be loaded. Otherwise, the settings and data will be updated in order to display the new statistic. In the case of a line chart, its summary text will be displayed. The object `TotalReader` will parse the `statistics_name_total.csv` into an array, which allows the display of these values in the summary. Changing the mode of a statistic does not require the settings to be reloaded or the summary text to be adapted, the data for the statistic is simply updated.

```
var statistics_name = new function () {
    this.getTitle = function () {
        return "Display name of Statistic";
    };
    this.getDescription = function () {
        return "Short description of the statistic. It will appear "
               + "below the display name in the menu.";
    };

    var totalReader = new TotalReader ("statistics_name");
    this.getTotalText = function () {
        return "Summary includes <b>" + sepThousands (totalReader.values[0])
               + "</b> absolute numbers or <b>"
               + totalReader.values[1] + "%</b> percentage values";
    };
    this.getModes = function () {
        return new Array ("daily", "monthly");
    };
    this.getType = function () {
        return "line";
    };
};
```

Listing 5.1: Statistics configuration

CHAPTER 6

Different Views into the Malware Landscape

In this chapter, we present in detail the charts that we developed with the intent to gain more insights into the operation of the Anubis platform and into common malicious behavior. In order to clearly structure the available statistics, we divide them into five different categories. In the following, we describe these five categories:

- The category **General** contains statistics about Anubis itself and its operation. For example, we look at the files submitted to Anubis. Here, we do not deal with the behavior of the executables but concentrate on evaluating the management data. This provides a detailed overview of the state of Anubis. The contained statistics are described in the Sections 6.1 to 6.10 in more detail. Most of the data used for these statistics is not available in the analysis reports of the individual samples but instead is stored in the database.

The remaining categories deal with the behavior of the samples we have analyzed. The data for these statistics originates from the analysis reports of the samples.

- The category **File Activity** includes statistics about the behavior of the executables related to file system interaction. Basic creation and deletion of files are examined, but we observe more complex behavior, like the modification of Windows system files, as well. We describe these statistics in the Sections 6.11 to 6.16.

- The category **Network Activity** covers the network behavior of analyzed samples. For example we draw conclusions about the most popular IRC servers or the number of samples sending spam e-mail. While most of this information is made available in the analysis reports, some protocols have been investigated in more depth. In these cases we rely on the more thorough analysis. These statistics are described in the Sections 6.17 to 6.27.

- In the category **Registry Activity** we examine basic registry interactions. These statistics are described in the Sections 6.28 to 6.29. More advanced registry interaction that causes alterations of system settings is not examined here, but in the next section.

- The category **System Interaction** contains statistics about the behavior of samples that alter system settings or perform some kind of process or service interaction. These statistics are described in the Sections 6.30 to 6.42.

We use two different kinds of charts to present the statistics: either line charts or pie charts. All the charts present their data in temporal dependence.

Our line charts show the information in relation to the time period that they occurred in. A line chart is loaded in its default interval mode[1] and its default time frame. The default time frame is twelve months for monthly statistics, 30 days for daily statistics and 48 hours for hourly statistics. We use the x-axis to represent a time period while the dependent data appears on the y-axis. Additionally, we offer the raw data in the form of CSV files for download. We display the date of the most recent update and the total values of the entire lifetime of Anubis are summarized below the chart. The user now has the possibility to change the interval used by the chart, which causes the chart to load the desired data.

We show at least two lines that represent different data or different perspectives on the data in each line graph. This favors direct comparison of the values. The user can interact with the chart. Selecting or deselecting the checkbox next to a graph's label causes the graph to appear or disappear. Clicking onto a label causes it to be underlined. A balloon that contains the value for the current date hovered over on the x-axis will be presented for each graph that has an underlined label. Additionally the values for the selected date will appear next to the labels if the user hovers over the chart with the mouse cursor. Finally the user can get a complete overview by clicking on the "Show all" link and zoom into a new time frame by selecting the desired period. A scrollbar enables the user to scroll horizontally, thus allowing for investigation of more recent or older values at the same zoom level.

Most line charts display graphs in different scales[2], one on the left y-axis and the other one on the right y-axis. In order to not confuse the user with an overloaded chart, we decided to show graphs for one scale only by default. The others are invisible by default but can be displayed on request at any time.

Since this kind of interaction cannot be performed on the following illustrations of the statistics, we made all graphs visible but point out the scale that applies to the graph and which graph would be invisible by default. Furthermore, we chose to present a monthly overview of the year 2009. If we wish to highlight properties of a statistic that cannot be seen in this overview, we provide additional illustrations.

Pie charts on the other hand are used when the underlying data does not allow the representation as line chart. This is typically the case when the label of a graph would have to change in different time periods. A pie chart is loaded for the current month. As in the line chart, the date of the most recent update is displayed. A menu above the pie chart lists all months for which data

[1] Most statistics use a daily interval as default.

[2] Percent and total numbers is the most common mixture.

is available and gives the user the opportunity to investigate past months. The pie chart requires one CSV file per month. All these CSV files are available for download. By hovering over the pie, the label and value for the examined slice are presented in a balloon. Additionally, each slice can be pulled out by clicking on the slice or its label. For the following illustrations we selected one month in 2009 at random.

It depends on the respective statistic's preferences, which kind of chart is shown. But since pie and line charts require different source data, it is not possible to show both charts for a given statistic.

6.1 Submissions

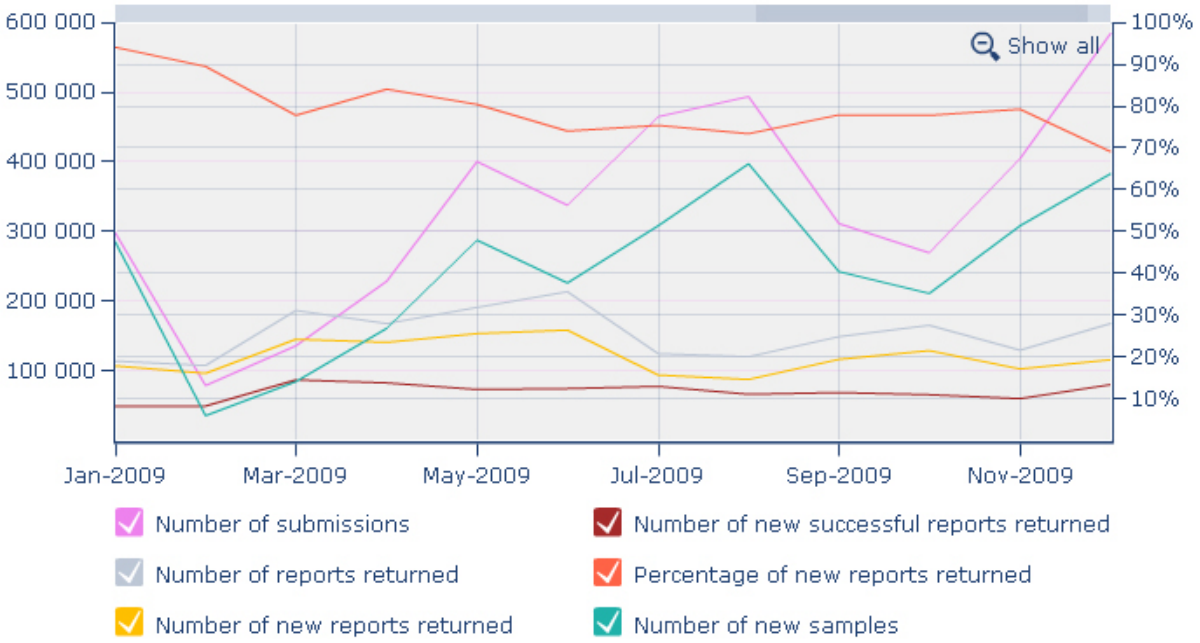


Figure 6.1: Monthly submissions

This chart illustrated in Figure 6.1 shows the amount of submissions to Anubis, the number of new samples submitted and the number of new, cached and successful reports returned along with its share of new reports in a monthly interval for the year 2009. This chart is available in daily or monthly intervals.

Anubis saves three different dates for each sample that was submitted. The *create time* represents the time at which the sample was successfully uploaded and added to the analysis queue. The time at which Anubis starts the analysis for the sample is saved as the *analysis start*. As soon as the analysis terminates, this time is saved as the *analysis end*. An individual analysis run for a file either ends successfully by delivering an analysis result or fails with a specific error code.

Additionally, we generate a checksum for each sample. It is saved along with the create time of the sample if the checksum does not exist yet. If the checksum of the sample is known already, we do not overwrite this time, thus this timestamp marks the date that we have *first seen* a sample.

- **Number of submissions**: This graph represents the number of samples that have been submitted to Anubis during the specified time interval, thus all their create dates lie in the same interval. Although this graph uses the same scale as the following three graphs, this graph is hidden by default because the user usually is more interested in the number of analyses rather than in the number of submissions, Moreover, this graph is prone to freak values, which might cause a distorted representation of the other graphs.

- **Number of reports returned**: This graph uses the end of the analysis as time to group the reports that have been returned and represents their number. Even if the analysis of a sample has already started on the previous day or in the previous month, the report is only returned at the end of the analysis and this it counts to the following time period.

- **Number of new reports returned**: This graph uses the same date for grouping as the previous graph - the time of the end of the analysis process. But only reports whose primary analysis subject has not been analyzed previously with the same version of Anubis are counted and are considered as new. On insertion, we assign a unique id to each task. The first thing that is checked upon submission is, whether a sample with an identical MD5 checksum has been analyzed before. If a sample is found, the new sample usually is not analyzed again, but the id of the previously analyzed sample is assigned as its result id. If no sample with an identical MD5 checksum can be found then the id of the sample itself is assigned as its result id, once the analysis process is finished. Thus, we can determine whether a report is new by comparing the sample's id to its result id. If they match, the report is considered as new.

- **Number of new successful reports returned**: This graph represents the number of reports that are new and whose analysis results in *exit code* 0. This number represents the samples that actually could be analyzed, thus consumed analysis time by one of the Anubis workers.

- **Percentage of new reports**: This graph visualizes the percentage of reports that are new. Since it uses a different scale than the other graphs, this graph is invisible by default.

- **Number of new samples**: This graph represents the number of samples that are new to Anubis. These samples are grouped by the create time of their first submission. It is related to the graph showing the *number of submissions* and is thus hidden as well.

Altogether, 5.5 million samples have been submitted to Anubis. 4.1 million of the samples are unique. 3.2 million reports have been returned, 2.6 million of these are new. So the percentage of new samples is over 80%. Of the new reports returned, more than 1.5 million could be analyzed successfully and resulted in exit code 0.

We present 5 graphs in this chart. We know that this can get confusing quickly. That is why we decided to make only the graphs on the numbers of reports returned visible by default.

The default view of the graph can be seen in Figure 6.2. These are the most important numbers since they represent the throughput of Anubis. The two graphs on the submissions to Anubis are presented seperately in Figure 6.3. In order to observe correlations between these graphs, they are all packed into this chart.

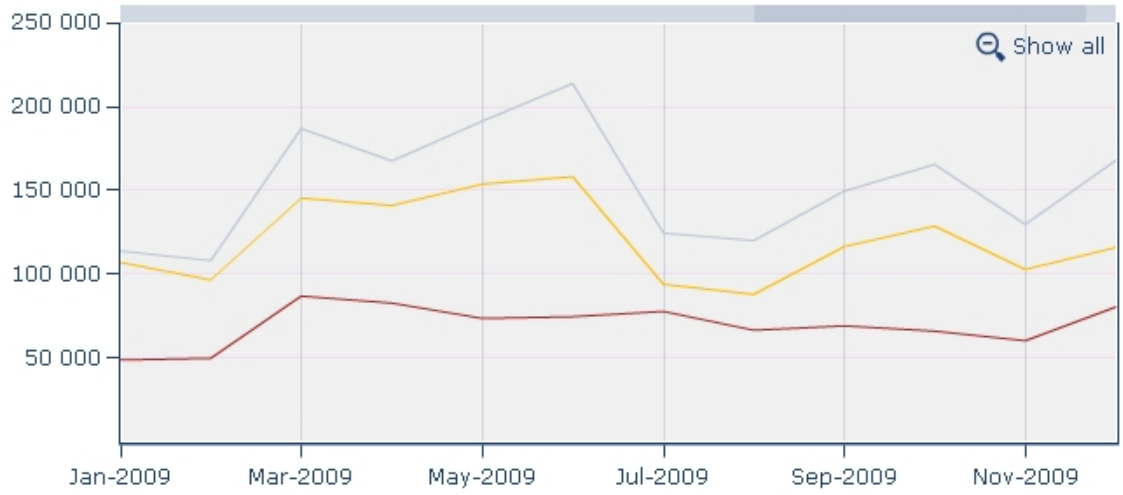


Figure 6.2: Monthly reports, new reports and new successful reports returned

In 2009 we can see that the percentage of new reports returned declines from 94.1% to 69.07%. This is possibly due the fact that the number of analyzed samples grows rapidly and it is thus more likely for an incoming sample to be known already. Due to Anubis' risen popularity it is also more likely that two submitters submit the same sample, even though it has not been in the wild for a long time. Additionally, the chart implies that a decline in the percentage of new reports returned results in an incline in analyzed samples. One fact that would support this assumption is the reduction in analysis time due to double submissions, as they do not require a new analysis. A cached report for this sample is returned instead. But double submissions are not the sole reason for an incline in reports returned. As mentioned above, a report is returned if an error occurs during analysis as well. We will investigate the correlation of the exit code and the amount of analyzed samples in Section 6.5. To see the number of samples that actually consumed analysis time, we introduced an additional graph that considers the exit code - the number of new successful reports returned. This is more stable than the other two graphs representing analysis numbers - it is almost constant in 2009.

The tremendous increase of submissions to Anubis can be explained by additional partnerships with major submitters that have been entered in 2009. In 2009 Anubis had four times as many submissions as in 2007 and 2008 combined. Finally, a constant extension and improvement of the analysis environment contributes to an increase of analyses. The difference between the number of submissions and the number of new samples is similar to the percentage of new reports. By these numbers, we can conclude that our report caching works efficiently, and a new report is generated for new samples in most cases.

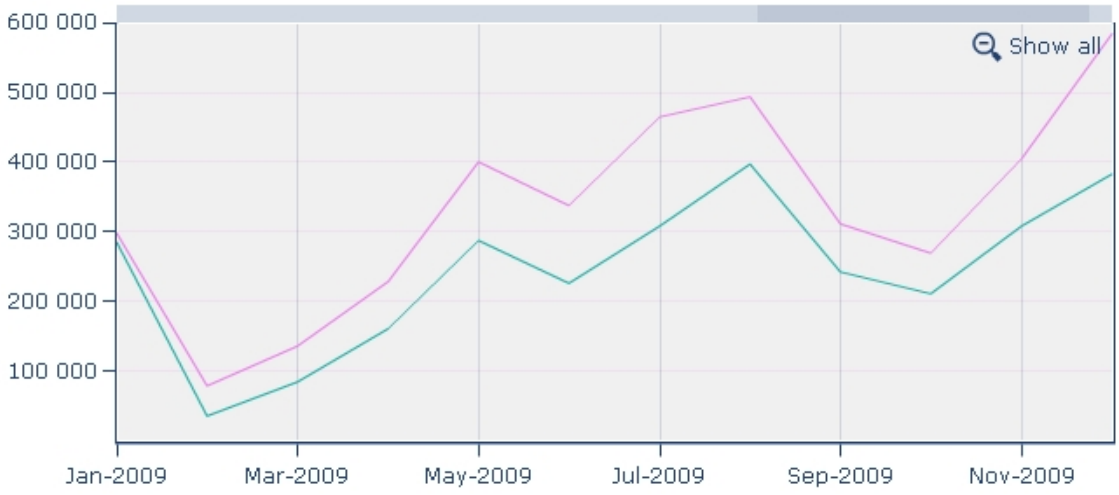


Figure 6.3: Monthly submissions and new samples

6.2 Manual Submissions

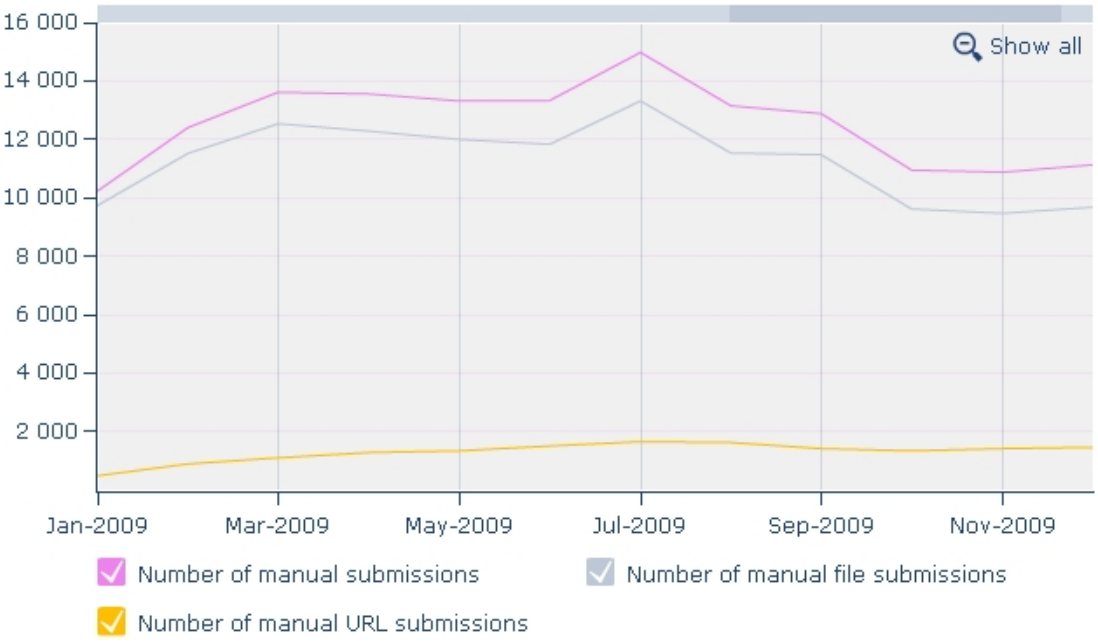


Figure 6.4: Monthly manual submissions

The chart presented in Figure 6.4 shows the amount of manual submissions to Anubis and their share of file and URL submissions in a monthly interval for the year 2009. This chart is available in daily and monthly intervals.

When a user submits a file via the web interface, he has the opportunity to solve a captcha. If the captcha has been solved correctly the sample receives a higher priority and will be analyzed more quickly. Since the solving of a captcha is not the only cause for a higher priority we cannot rely on the value of the priority to classify a submission as manually submitted. The correctness of the captcha is a better indicator although we miss a few manual submissions where the captcha has not been solved correctly.

Since all graphs in this statistic use the same scale, they are all displayed by default. Because the chart displays two graphs that add up to the third we saw no reason for an additional representation in percentage. We use a sample's create timestamp as time for grouping.

- **Number of manual submissions**: This graph represents the number of samples that have been submitted with a correctly solved captcha.
- **Number of manual file submission**: This graph gives the share of file submissions. In this case the behavior of the submitted executable is monitored.
- **Number of manual URL submissions**: This graph visualizes the share of manual URL submission. In this case a URL is submitted to Anubis instead of a file. Anubis loads the URL with Microsoft's Internet Explorer and monitors the behavior of the Internet Explorer.

In total, 176000 samples have been submitted to Anubis manually. 16000 of those are submissions of URLs and in 160000 cases a file has been submitted for analysis. In 2009, 5% to 10% percent of the samples analyzed by Anubis are submitted manually. We can see that Anubis' primary analysis type - the file analysis - is clearly preferred. Over 90% of manual submissions perform a file analysis. Since almost all URLs are submitted manually, we did not bother to perform this distinction of file and URL analysis in the previous statistic.

Besides the numbers of manual submissions, we are also interested in the quality of manual submissions. Taking a closer look at the URL submissions, we noticed that a considerable amount (20%) of the URLs has an `.exe` postfix. In these cases, users probably intended a file analysis but submitted the URL to the file rather than the file itself. The quality of manual file submissions is surprisingly high. Manual file submissions only have a share of 14% of invalid executables. The overall percentage of invalid executables, which is further discussed in Section 6.5, is higher.

6.3 Reports per Worker

The chart presented in Figure 6.5 breaks down the number of new reports returned for their workers. This chart can be viewed in daily and monthly intervals. To group the samples, we used their analysis end. The analysis start of a sample would do just as fine, but using the analysis end makes the generation of the statistic fast, as we would need one join more in the query using the timestamp of the analysis start.

We decided not to show a graph for each worker, because 64 different lines would overload the chart. Therefore, we decided to show only groups of workers. Each label in Figure 6.5 stands for one group. These groups might run different versions of Anubis and run at different geographical locations.

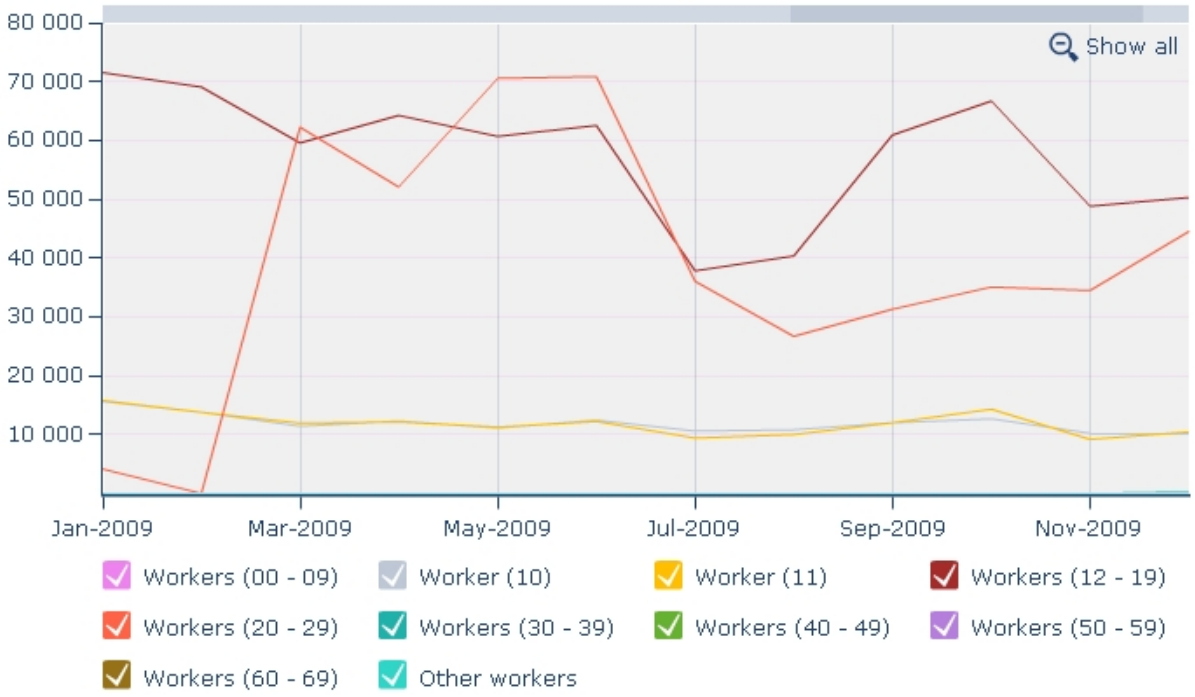


Figure 6.5: Monthly reports per worker

Each worker has a unique name. This name ends with the ID of a worker (e.g. 'worker-10'). This ID represents the last byte of the worker's IP address. We extract the IDs from the workers' names and use it to group the workers.

- **Workers (00 - 09)**: This graph represents the number of new reports returned by workers 00 to 09. These workers performing analyses from October 2007 until September 2008 and have been replaced by other workers since then.
- **Worker (10)**: This graph shows the number of new reports returned by worker 10. Worker 10 deserves its own group because it runs a different version of Anubis to test new features.
- **Worker (11)**: This graph gives the number of new reports returned by worker 11. Like worker 10, this worker too tests new versions of Anubis.
- **Workers (12 - 19)**: This graph stands for the number of new reports returned by workers 12 to 19. These workers run at the Vienna University of Technology (TU Vienna).
- **Workers (20 - 29)**: This graph corresponds to the number of new reports returned by workers 20 to 29. Like the previous group of workers, this one runs at the TU Vienna.
- **Workers (30 - 39)**: This graph describes the number of new reports returned by workers 30 to 39. These workers are located at the Institute Eurecom in Riviera (Eurecom) and run since January 2010.

- **Workers (40 - 49)**: This graph shows the number of new reports returned by workers 40 to 49. These workers are located at the Eurecom and run since January 2010.

- **Workers (50 - 59)**: This graph represents the number of new reports returned by workers 50 to 59. These workers are located at the Eurecom and run since February 2010.

- **Workers (60 - 69)**: This graph gives the number of new reports returned by workers 60 to 69. These workers are located at the Eurecom and run since February 2010.

- **Other workers**: This graph visualizes the number of new reports returned by other workers. A samples worker has only been saved since October 2007. Analyses that finished prior to this date have no worker assigned and are thus summarized in this group.

In Table 6.1, we list the number of new reports returned for each worker group. The groups that returned the most new reports are the groups consisting of workers 12 - 19 and workers 20 - 29. This is due to the fact that these groups were around the longest. The groups located at the Eurecom only started their work in 2010 and are part of Anubis' expansion. Figure 6.6 illustrates the launch of workers 50 - 59 and workers 60 - 69 on February 23rd 2010.

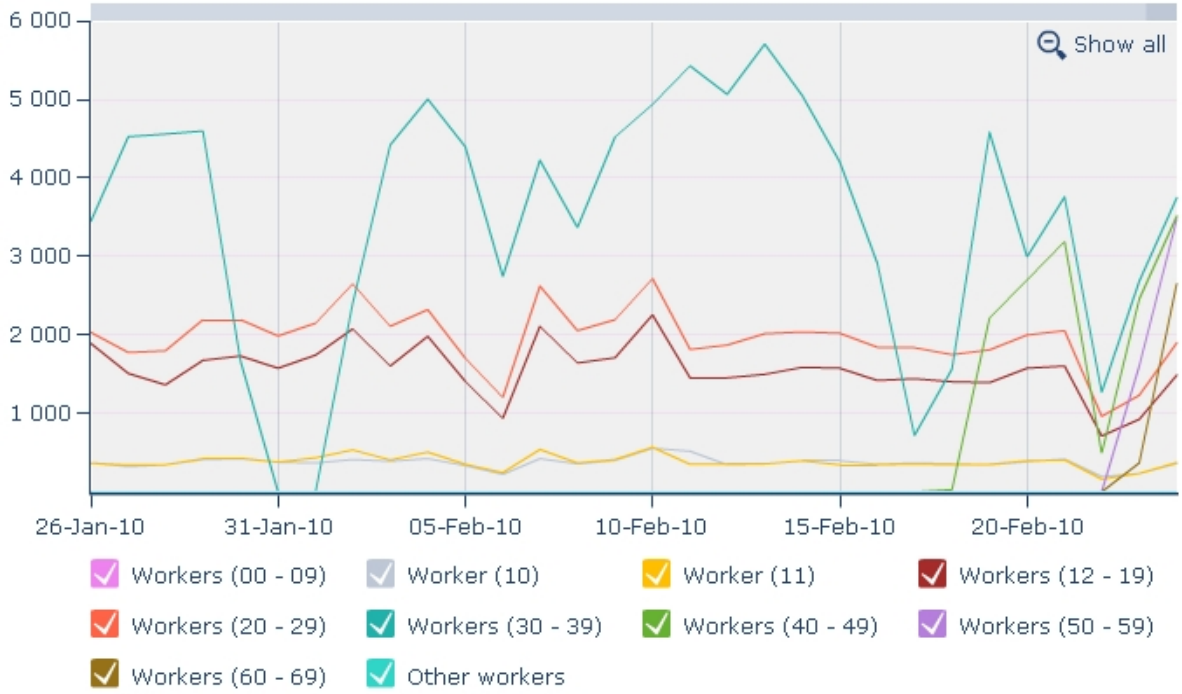


Figure 6.6: Daily reports per worker

The main cause of different numbers of reports returned by the groups is their different number of running workers. The number of workers running in a group reaches from one to ten.

Group	New reports returned
Workers 00 - 09	674 990
Worker 10	23 512
Worker 11	193 561
Workers 12 - 19	1 040 774
Workers 20 - 29	697 800
Workers 30 - 39	163 603
Workers 40 - 49	39 220
Workers 50 - 59	5 915
Workers 60 - 69	3 146
Other workers	86 428

Table 6.1: Number of new reports returned per worker group

6.4 Worker on Duty

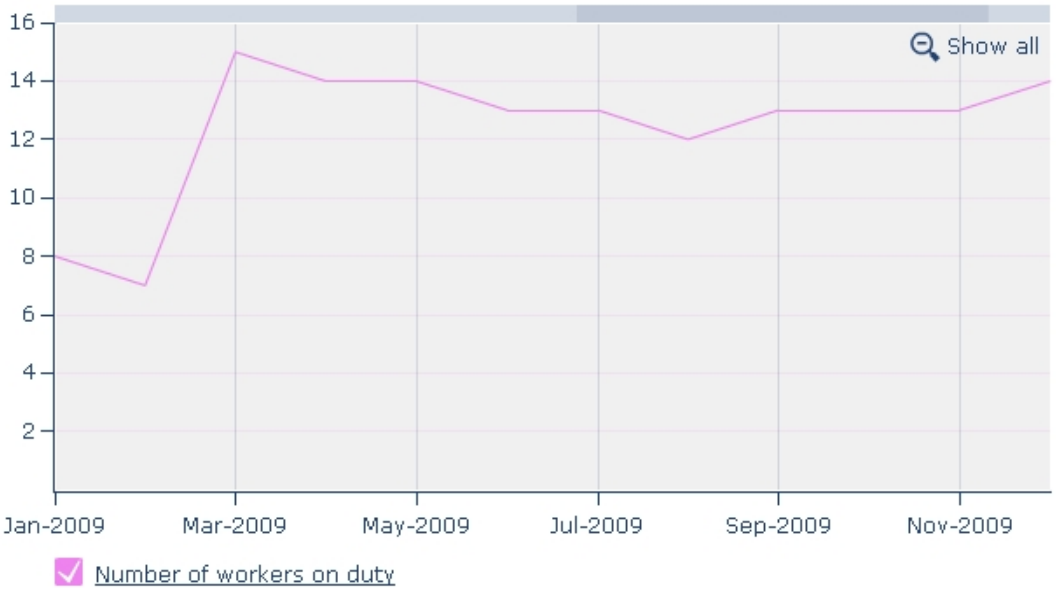


Figure 6.7: Monthly Worker on Duty

The chart in Figure 6.7 gives an overview on the number of workers that analyze samples in 2009. This chart is available in monthly and daily intervals. We use a sample's analysis end to get the group affiliation of a worker.

- **Number of workers on Duty**: This graph represents the number of different workers that were spotted analyzing samples.

In 2009, we had an average of 12.42 workers analyzing samples submitted to Anubis. Altogether, we spotted 63 different workers. At the beginning of 2010 we started further expansion of Anubis and added more workers. We managed to increase the number of workers in January 2010 to 27 and in February to an all-time high of 49.

6.5 Exit Codes

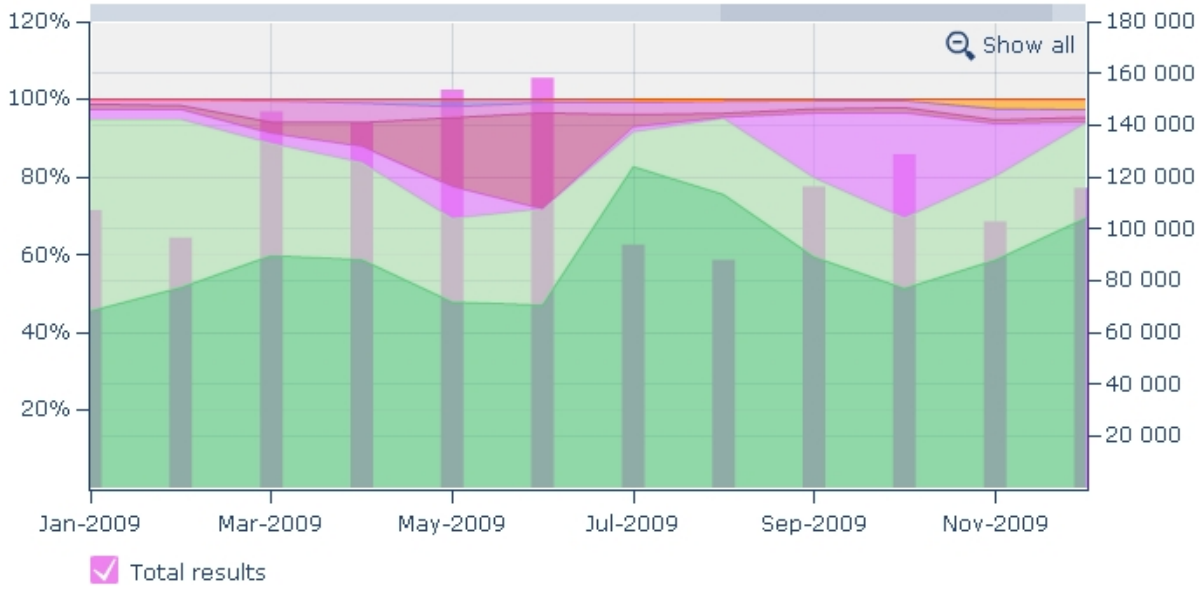


Figure 6.8: Monthly exit codes

The chart presented in Figure 6.8 shows the monthly exit codes produced by the analysis process in a stacked line chart for 2009. It is available with daily and monthly intervals.

Since we distinguish eight exit codes for this chart, it has not shown practical to list all 9 labels beneath the chart. It simply exceeds the space available for labels, if the chart is not viewed in a full screen sized window. Therefore, we decided to make the labels available in the balloons, once the user hovers over a date with the mouse cursor. These labels can be seen in Figure 6.9 for January 9th 2010. In this chart we examine the following exit codes:

- **Exit code 0**: This exit code is returned if the analysis for an executable is completed successfully.
- **Exit code 7**: This exit code refers to an invalid executable that has been submitted for analysis. We examine the header of each submitted sample and start the analysis process only for samples that could be identified as Windows executable. This error is the only one that does not reflect erroneous behavior of Anubis and cannot be influenced by Anubis, since we do not have control over the samples submitted to Anubis.

- **Exit code 24**: This exit code is generated by `analyze.py` when an analysis report cannot be found.
- **Exit code 30**: This exit code indicates an error that occurred in one of the scripts called by `analyze.py`. These scripts perform post analysis tasks like the analysis of network traffic or the generation of the report summary.
- **Exit code 31**: This exit code occurs when Qemu crashes and is logged by `analyze.py`. Qemu is the virtual machine that hosts the Windows environment in which the analysis is performed.
- **Exit code 100**: If analysis exceeds the timeout of usually eight minutes, the analysis process is killed and `analysis_server.py` returns exit code 100.
- **Exit code 1000**: `analysis_server.py` returns this exit code when an error occurs during the quick analysis of a file. The quick analysis includes the determining of executable and non executable files.
- **Other exit codes**: Other exit codes include 20, 21, 22 and 23. These exit codes are raised by `analyze.py` and are returned if it is called with erroneous arguments or the analysis subject cannot be found. These are severe errors but occur very rarely and are thus not listed individually.
- **Total results**: This number represents the absolute number of analyses for the interval as a bar graph in the background. This number always represents 100%. Although this graph can be told well apart from the other graphs, it is hidden by default because it uses a different scale than the exit code distribution in percent.

Exit code	Percentage
0	60.09%
7	20.72%
24	4.62%
30	5.29%
31	3.59%
100	0.86%
1000	0.48%
other	4.35%

Table 6.2: Distribution of exit codes produced by Anubis

Inspecting the distribution of the error codes the relatively high percentage of error code 30 in May and June 2009 stand out. This might be due to a bug introduced to one of the post analysis scripts that has been fixed later. A similar explanation can be given for the high values of exit code 24 in October 2009. Generally, this chart allows one to get an overall impression on the health of Anubis' analysis environment.

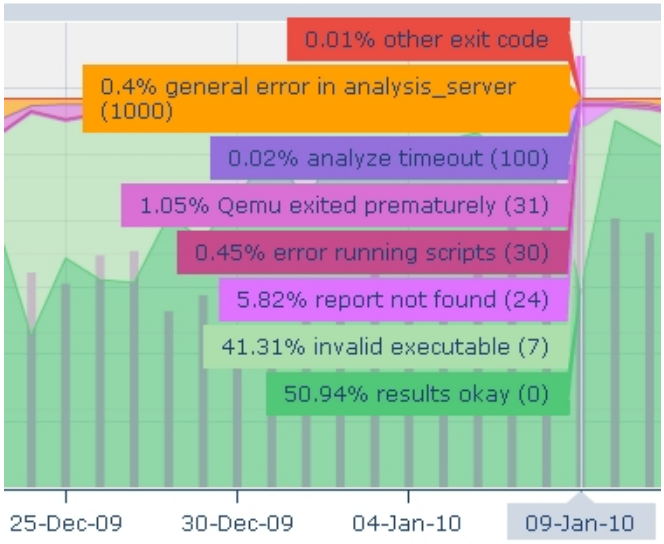


Figure 6.9: Exit codes for January 9th 2010

Additionally, peak values of analyses can be explained. In Figure 6.9, we see that the number of analyses on January 9th 2010 is very high. Further investigations show that this is due to the high share of invalid executables submitted on that particular day. Invalid executables do not require a full analysis, as they are identified in the quick analysis already and thus consume less time.

The distribution of the error codes of all 2.6 million analyses is listed in Table 6.2 and shows that Anubis performed correctly in over 80% of the cases.

6.6 Errors per Worker

The chart in Figure 6.10 describes the percentage of errors that occurred in each group of workers in 2009. This chart is available in daily and monthly intervals. The percentage is calculated by dividing the number of new reports that return a different exit code than 0 or 7 with the total number of new reports for each worker group in an interval. A sample's analysis end is used to determine the interval a sample is assigned to.

In Section 6.3, we gave a detailed description of the worker group composition. The same groups are used in this statistic.

Table 6.3 lists the overall percentage of errors produced by each worker group. Interestingly, the test workers do not have a significantly higher error rate than the other workers running at the TU Vienna. It is noticeable that the four groups of workers running at the Eurecom have a significantly lower error rate than the remaining groups. This is probably due to the fact that these workers started their task only recently and have not experienced a transition to a new version of Anubis. These transitional periods are usually the main cause of errors until all the little tweaks and bugs are fixed and everything goes back to normal. Since Anubis is improved continuously it

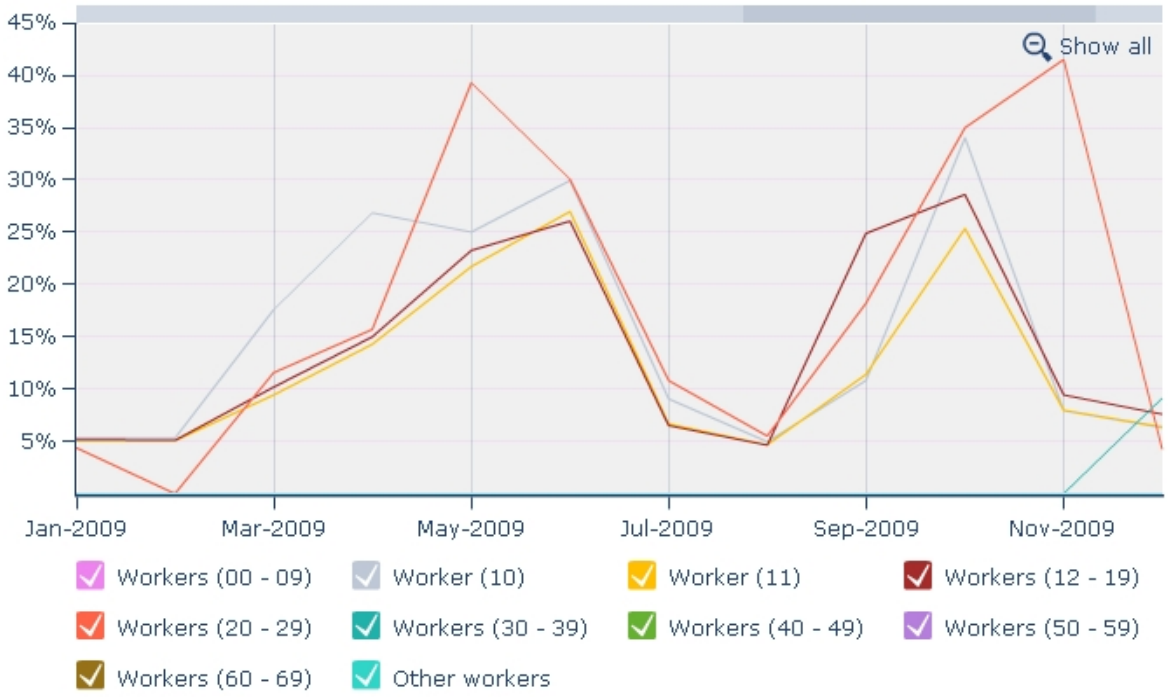


Figure 6.10: Monthly Errors per Worker

is understandable that the worker groups that operated in the early days of Anubis are most prone to errors.

Group	**Location**	**Percentage**
Workers 00 - 09	TU Vienna	24.38%
Worker 10	TU Vienna	15.50%
Worker 11	TU Vienna	11.68%
Workers 12 - 19	TU Vienna	13.49%
Workers 20 - 29	TU Vienna	18.86%
Workers 30 - 39	Eurecom	2.96%
Workers 40 - 49	Eurecom	6.64%
Workers 50 - 59	Eurecom	4.83%
Workers 60 - 69	Eurecom	4.61%
Other workers	TU Vienna	39.42%

Table 6.3: Percentage of errors for each worker group

6.7 Analysis Delay

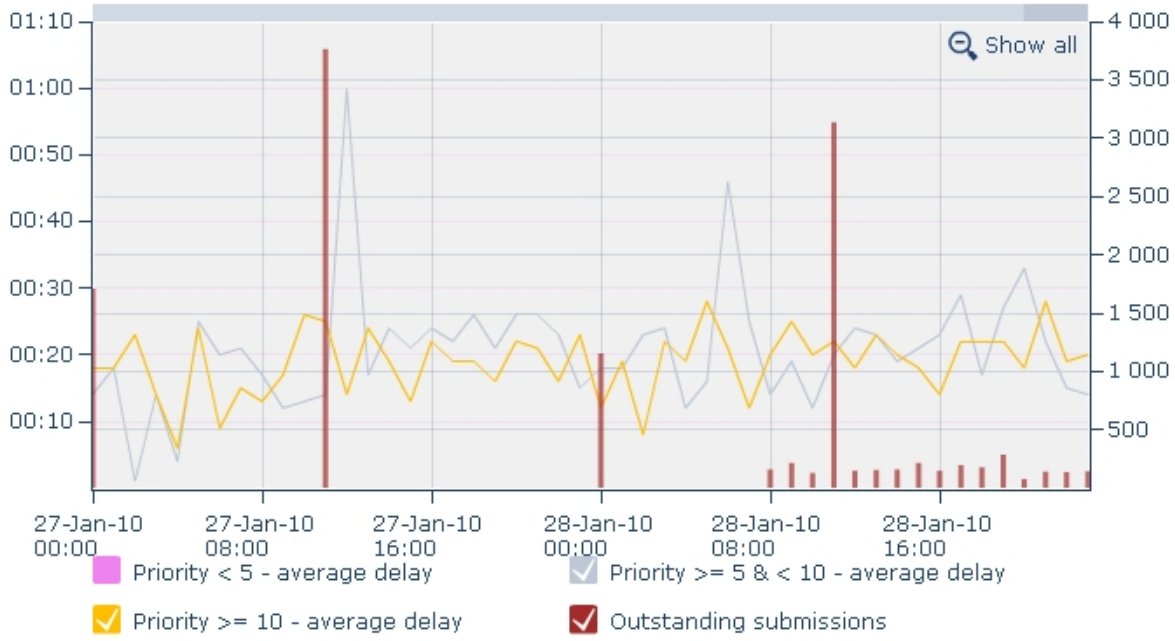


Figure 6.11: Hourly analysis delay

The analysis delay is illustrated in Figure 6.11 and describes the average waiting time of submitted samples until their analysis is started for the past 48 hours. The waiting time is equal to the difference between the *create time* and the *analysis start* time. The grouping of the samples occurs according to their create time. The samples are distinguished by their priority, which is the main influence on the delay. This chart is available in an hourly and a daily mode. The hourly chart goes back one month while the daily chart covers the past 3 months. This chart is the only one that is not complete for the history it represents. A sample that was submitted a week ago and whose analysis started only now might change the average delay of submissions seven days ago. This is the reason for the limited period of time this chart goes back in history.

- **Priority <5**: This graph shows the average delay for submissions with a priority less than 5. This covers batch processes. Since the average delay for these submissions is often greater than a day, we hide this graph by default. That way, the curves of the other graphs can be recognized and they do not appear as a straight line at the bottom of the chart.
- **Priority >= 5 & <10**: This graph lists the average delay of submissions with a priority between 5 and 9. Manual submissions reside in this category. We aim to keep this value low, as users who submit a sample manually need a quick response.
- **Priority >10**: This graph lists the average delay of submissions with a priority greater or equal to 10. Privileged users are in this category. These users usually submit a low number

of samples, but require a very urgent response. We aim to give this submission to the next worker that finishes his current task.

- **Outstanding submissions**: This number gives the number of tasks that have been submitted but not analyzed yet. Thus, they are waiting for analysis. This graph is presented in bars. It can be well distinguished from the other line charts and it is displayed by default even though it uses another scale.

This chart reflects the load of Anubis. The batch submissions are analyzed when no manual submissions are waiting for analysis. They usually have to wait at least one day in average before they are analyzed. Since the graph for batch submissions is not that interesting we have hidden it in Figure 6.11 in order to have a better view on the other graphs.

Response time for manual submissions is more critical. While we can see load issues on January 4th and 7th 2010, we usually manage to keep the delay below 10 minutes for submissions with a priority between 5 and 9. For submissions with a priority above 9 we managed to keep the average delay well below one minute.

The outstanding submissions give an overview on the age of the queue waiting for analysis. Usually submissions do not have to wait longer than one week before they are analyzed. To avoid unproductive periods, we always try to keep at least 20000 submissions in queue.

6.8 File Types

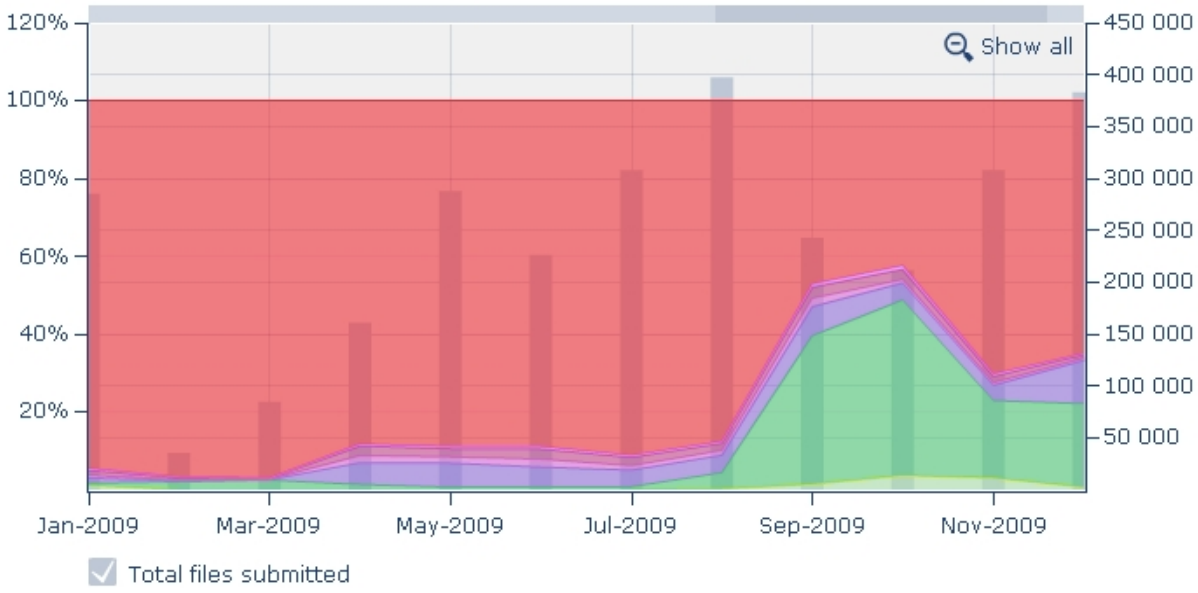


Figure 6.12: Monthly file types

The chart in Figure 6.12 gives an overview on the type of the files that were submitted to Anubis in 2009. It is available in monthly and daily intervals. The files are grouped by the date of their first occurrence.

This chart is structured similarly to the chart representing Anubis' exit codes, which is described in Section 6.5. In this chart, the number of labels exceeds the space available for labels as well. That is why we chose to make the labels available as balloons only, once the user hovers over a date with the mouse cursor. These labels can be seen in Figure 6.13 for January 2nd 2010. In this chart we examine executable and non executable file types. The graphs representing the portable executable (PE) format are coded in a green shade while the graphs for the non executable types are colored in pink and purple shades.

In the quick analysis we investigate the fields of PE files and save these to our database. Additionally we save the mime type of every file.

We distinguish the following file types:

- **PE DLLs**: This graph represents the percentage of files that are a PE file, and that have an according entry in the PE file table. Furthermore the image characteristic `IMAGE_FILE_DLL` has to be set for these files.
- **PE drivers**: This graph gives the percentage of PE drivers. A PE driver may not have the image characteristic `IMAGE_FILE_DLL`. Additionally, it has to be of the subsystem `IMAGE_SUBSYSTEM_NATIVE`.
- **PE executables**: The PE executable is the most common PE file submitted to Anubis. For a PE executable the image characteristic `IMAGE_FILE_DLL` may not be set and it must be of one of the subsystems `IMAGE_SUBSYSTEM_WINDOWS_GUI` and `IMAGE_SUBSYSTEM_WINDOWS_CUI`.
- **HTML files**: This graph shows the percentage for files, whose mime type starts with `text/html`. We ignore the specific character set of the HTML file.
- **Text files**: This graph represents the percentage of plain text files. The mime type of these files starts with `text/plain`.
- **RAR files**: The percentage of RAR files represents the share of files, whose mime type starts with `application/x-rar`. Probably the user intended an analysis for the content of the RAR archive, but Anubis is only capable of unpacking ZIP archives.
- **ZIP files**: This graph gives the percentage of ZIP files submitted to Anubis. A ZIP file is identified by comparing the beginning of a files mime type to `application/zip` or `application/x-zip`. Although, Anubis is able to unpack a ZIP archive for analysis, only ZIP archives that are secured with the password 'infected' actually are unpacked if the user chooses this option in the web interface.
- **Other files**: This graph represents the percentage of all other files submitted to Anubis. Flash applications, audio and video files are common file types that are summarized in this category.

- **Total files submitted**: This bar graph gives the absolute number of files that have been submitted to Anubis in the given time period. This number represents 100% for the given time period. It is not visible by default, as it uses a different scale.

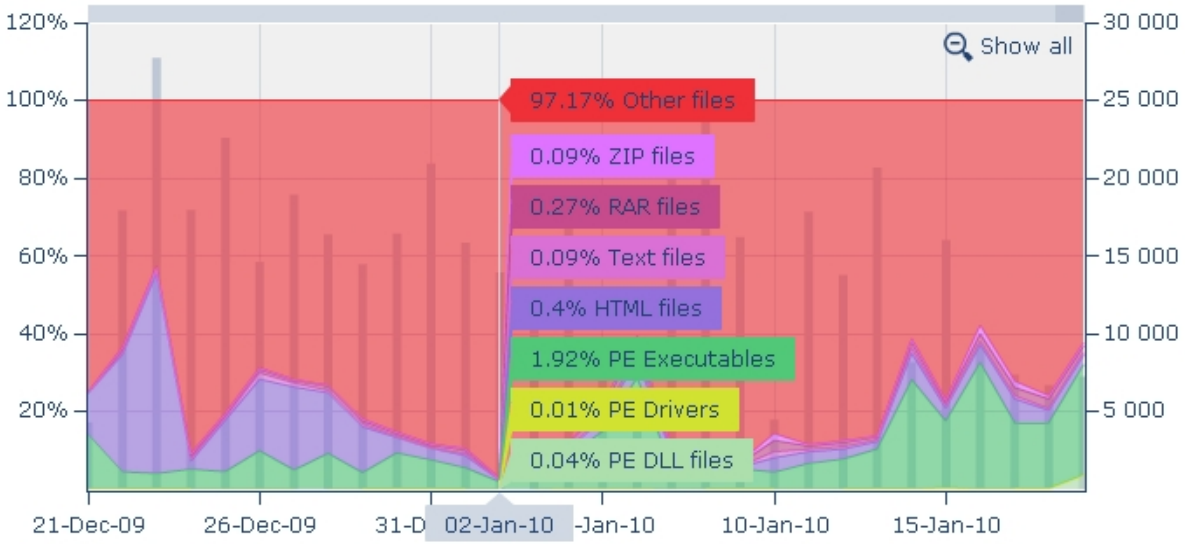


Figure 6.13: Distribution of file types for January 2nd 2010.

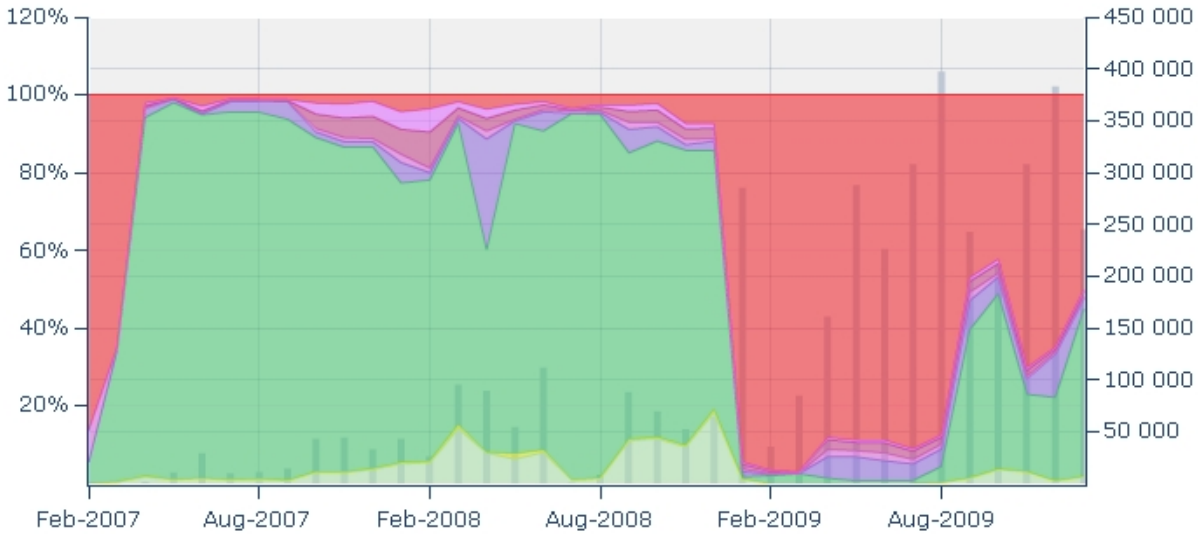


Figure 6.14: Monthly distribution of file types since the launch of Anubis.

Figure 6.12 reveals an alarming high percentage of not categorized file types. In Figure 6.13 we can even see a percentage of more than 97% of uncategorized file types and only 2% of PE files that can be analyzed by Anubis. This makes no sense since the submissions chart lists almost

5000 new successful analyses for January 2nd 2010, which is by far more than the 283 PE files stated in Figure 6.13 for this day. Taking a closer look at the mime types of the uncategorized files, we found that most files are of mime type `application/binary`, which is an executable indeed in most of the cases. Figure 6.14 gives a complete overview of the distribution of file types since the launch of Anubis in February 2007. The drastic decrease of PE files in January 2009 stands out. We suspect a bug in the determining of PE files that has been introduced sometime around the turn of the year. We are currently working on this issue and will update the statistic as soon as this bug is fixed and the information on the PE files is inserted into the database.

A total of 4.07 million files have been submitted to Anubis. Although the percentages of PE files and of other file types as a consequence are erroneous we list the current results of this statistic in Table 6.4. Once we fix the bug and update the PE information for our files, we expect to have less than 10% of the files in the category "other file types".

File type	**Percentage**	
PE DLLs	2.67%	31.45%
PE drivers	0.14%	
PE executables	28.64%	
HTML files	5.20%	68.55%
Plain text files	1.09%	
RAR archives	2.06%	
ZIP archives	0.88%	
Other file types	59.32%	

Table 6.4: Distribution of file types of files submitted to Anubis

6.9 Packers

The chart in Figure 6.15 describes the distribution of the most common binary packers for April 2009. The top ten packers are calculated for each month. Clicking on another month in the top menu loads the top ten packers for the given month.

A packer is used to compress or encrypt PE files. This makes it more difficult to reverse engineer these binaries. Furthermore, some packers like the 'Allaple Polymorphic Packer' produce different byte code in every run. By leveraging polymorphic packers, malware is capable of completely altering its code after every infection. Antivirus systems usually use signatures to detect malware, but as a consequence of an altered code, the signature changes too.

We use the program SigBuster to determine the packer used by a binary. The information generated by SigBuster is also made available in each analysis report in the general section. For the generation of the monthly top ten packers, we ignore the exact version of the packer and group all versions into a single category.

"Allaple Polymorphic Packer" and "UPX" are the most popular packers. They are part of the top 5 in the entire observation period. It stands out that most of the binaries, submitted to Anubis

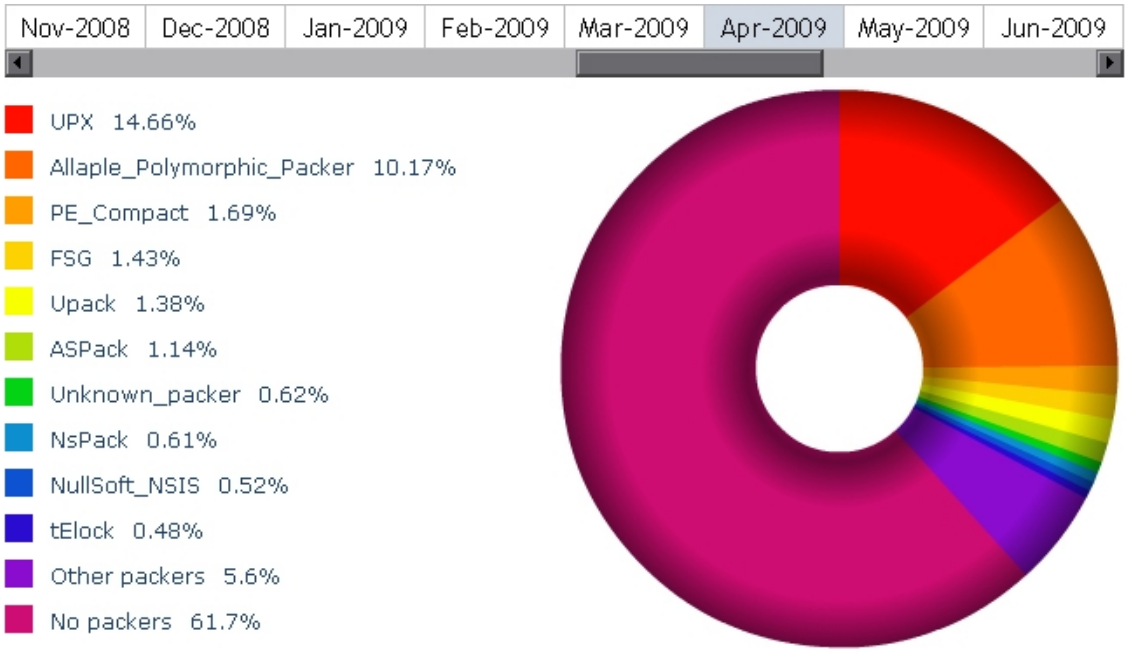


Figure 6.15: Packers in April 2009

are not packed at all. Since other studies show that more than 80% of malware is packed [41], we can conclude that we do not detect all packers reliably.

6.10 Virus Labels

Figure 6.16 describes the top ten virus labels for May 2009 according to the Ikarus Anti Virus scanner. This chart is available for every month. The month can be switched by clicking on its label.

We use several antivirus products to scan and label our samples. Currently, we use products by Ikarus, McAfee, Avira, Kaspersky and BitDefender. The files are scanned at the beginning of their analysis. Since we include the results of in our analysis report as well, we use their labels to produce the top ten of virus labels. To scan our files with even more antivirus products, we plan to submit our samples to VirusTotal [21]. VirusTotal performs antivirus scans with 40 different antivirus products.

Usually, less than 50% of the samples are identified as being malicious by Ikarus. Of course, not all the samples submitted to Anubis are actually malware. But since we receive zero-day samples to a fair share, antivirus vendors have not updated their signature database at the time of analysis and thus cannot identify the sample yet. This fact speaks in favor for the quality of our samples. Generally, we and the antivirus vendors receive the same samples contemporaneously. By using the service of VirusTotal we could request a repeated scan of our sample at a later point in time, for example three months after its initial scan, and observe the changes.

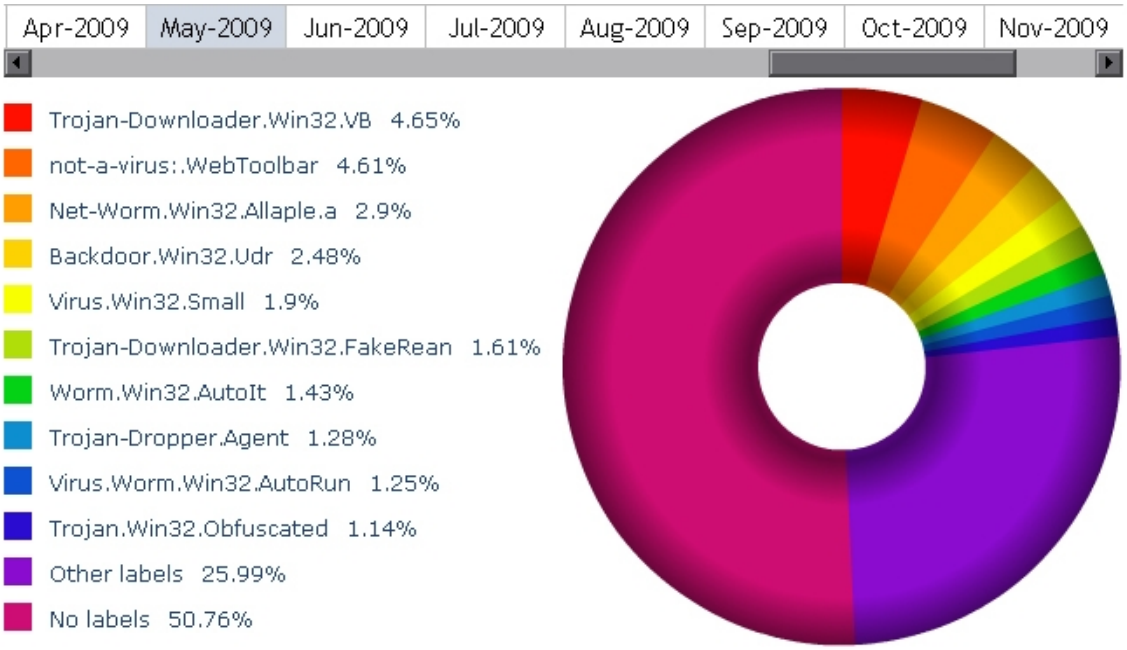


Figure 6.16: Virus labels in May 2009

Antivirus vendor	Percentage of viruses identified
Avira	77.44%
Kaspersky	76.52%
McAfee	68.18%
BitDefender	49.05%
Ikarus	44.88%

Table 6.5: Percentage of viruses identified by each antivirus vendor

In total we scanned 2.6 million files for viruses. In Table 6.5 we list the ranking of the antivirus products. The high percentage of identified viruses by Avira and Kaspersky can be explained by their superior identification of polymorphic viruses [44]. In Table 6.6 we list the percentage of samples that have been identified as a virus by any number of antivirus products. 90.49% of the scanned samples have been identified as virus by at least one antivirus product. We assume that this percentage includes false positives to some share, but 70.03% of our samples are identified by at least three antivirus products. This might be a more realistic number.

6.11 File Creation

The chart in Figure 6.17 provides information on samples that perform file creation in 2009. This chart is available in daily and monthly intervals. The date of the end of the analysis is used to count the samples and file creation tasks.

Number of antivirus products	Percentage of viruses identified
5 (all products)	20.64%
4	34.80%
3	14.59%
2	9.43%
1	11.03%
0 (no product)	9.51%

Table 6.6: Percentage of viruses identified by antivirus products

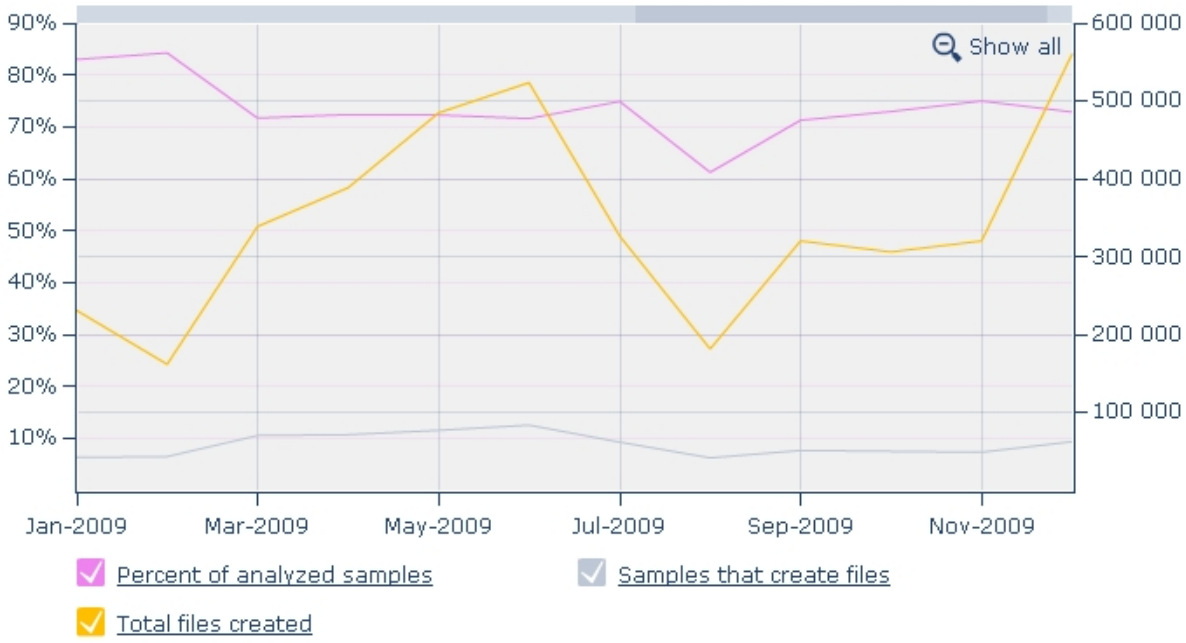


Figure 6.17: Monthly file creation

- **Percentage of analyzed samples**: This graph represents the share of analyzed samples that perform file creation.
- **Samples that create files**: This graph gives the absolute number of samples that create files. By default this graph is hidden, since it uses absolute numbers instead of percent.
- **Total files created**: This graph shows the number of file creations that have been performed. It is quite common that a single sample creates more than one file. Like the previous graph, this one is hidden by default as well.

Anubis monitored 7.9 million file creations by 1.2 million samples. On average a sample, which performs file creation, creates 6.6 files during analysis. 71.05% of the analyses create at least one file. This is not surprising because the creation of files is one of the most basic operations

a program can perform. Even malware creates files - as is demonstrated by our numbers - for example to copy themselves to the Windows directory.

File extension	Percentage of created files
no extension	21.23%
`.exe`	16.90%
`.tmp`	10.07%
`.dll`	5.61%
`.ini`	2.99%
`.txt`	2.73%
`.gif`	2.67%
`.lnk`	2.46%
`.bat`	2.40%
`.htm`	2.11%

Table 6.7: Percentage of extensions for created files

We do not examine the mime/type of files created, but the file extensions listed in Table 6.7 give an idea of the file's type. 21.23% of the files created do not have any extension at all. Along with the files that have a `.tmp` extension, they are most probably temporary files. Another 22.51% of the files are `exe` or `dll` files. These files are likely to be copies of the malware's program code in order to persist in the infected system.

6.12 File Deletion

The chart in Figure 6.18 visualizes the samples that delete files on a monthly basis in 2009. A daily view is provided for this statistic as well. The number of file deletions is calculated according to the sample's analysis end.

- **Percentage of analyzed samples**: This graph shows the percentage of analyzed samples that delete files.
- **Samples that delete files**: This graph represents the number of samples that perform file deletion. It is hidden by default.
- **Total files deleted**: This graph represents the total number of file deletions that have been detected by Anubis. Of course, more than one file deletion per sample is possible. This graph is not visible by default.

Overall, 0.7 million samples delete 2 million files. That makes 2.9 file deletions for 42.05% of the samples on average. The number of deleted files rarely exceeds the number of created files that have been examined in Section 6.11. Table 6.8 shows the distribution of file extensions of deleted files. Most files deleted are temporary files. Most probably, these files have been created by the sample previous to their deletion. Interestingly enough, 13.71% of the deleted files are

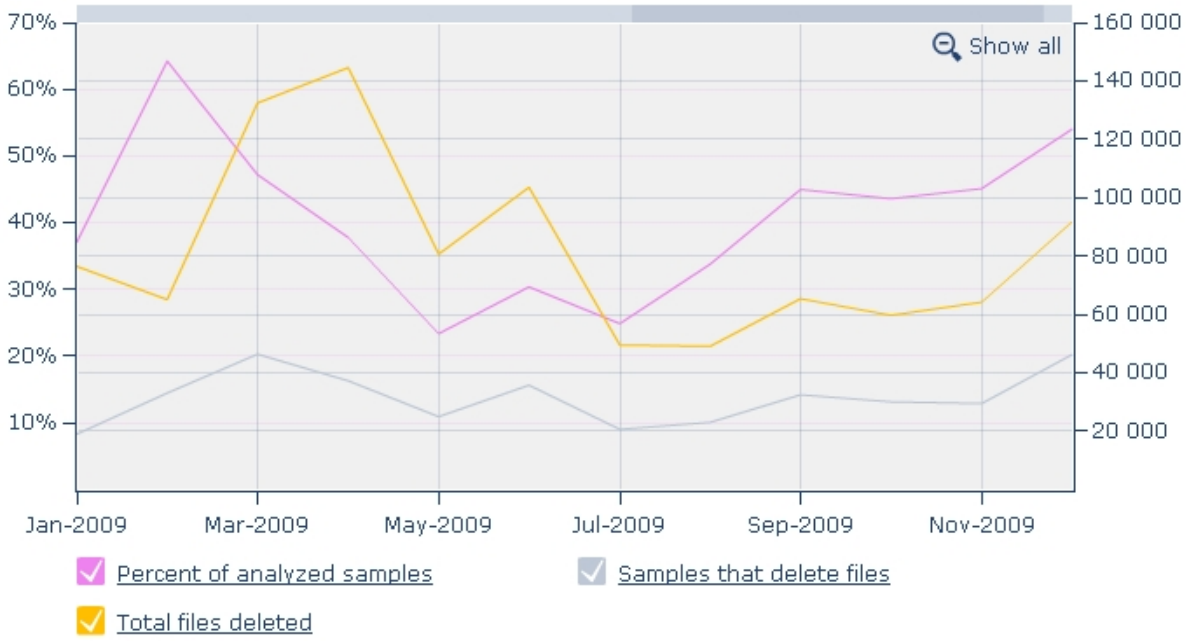


Figure 6.18: Monthly file deletion

File extension	Percentage of deleted files
`.tmp`	15.94%
`.txt`	15.59%
`.exe`	13.71%
no extension	5.67%
`.gif`	4.43%
`.bat`	4.42%
`.dmp`	3.16%
`.ico`	2.70%
`.epk`	2.15%
`.zk`	2.12%

Table 6.8: Percentage of extensions for deleted files

`exe` files. This file extension usually marks an executable. We think that malware whose only purpose it is to download additional malware might remove itself after it has performed its job.

6.13 File Modification only inside the User Directory

The chart presented in Figure 6.19 describes the samples that abstain from tampering with files outside the user directory in 2009. These samples do not require administrator privileges

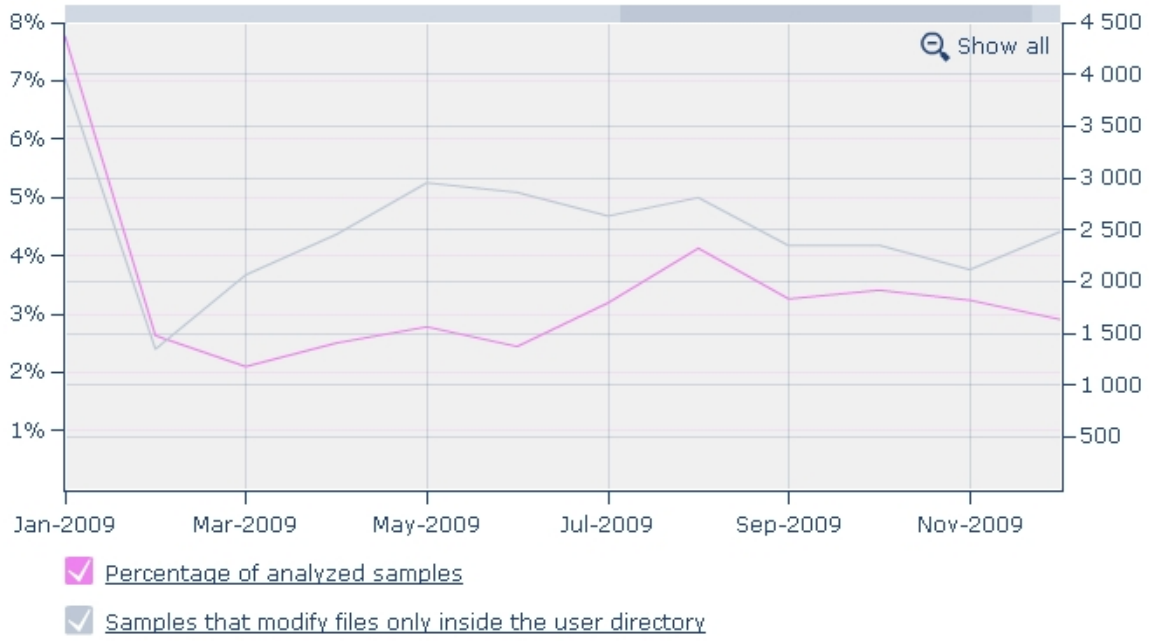


Figure 6.19: Monthly file modification only inside user directory

to perform modifications on the file system. Since other actions taken by a sample, like the installation of files or the modification of the registry might require administrator privileges, these numbers only give an upper bound. These samples are grouped by their analysis end either in monthly or daily intervals.

Directory name
`%userprofile%\`
`C:\DOCUME~1\`
`C:\Documents and Settings\`

Table 6.9: Locations of the user directory in Windows XP.

Basically, we want to discover write operations performed by the sample. If all these write operations occur inside the user directory, the sample would not require administrator privileges to perform them. A sample can perform a write operation by creating a file or a link, modifying a file or deleting a file. We count the number of total write operations and write operations to files, whose path begins with one of the directories listed in Table 6.9. If these numbers match we identified a sample that writes only to the user directory.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that modify files in the user directory only.

- **Samples that modify files only inside the user directory**: This graph gives the absolute number to the percentage of the previous graph and is thus not visible by default.

Only 56 000 samples perform their write operations only inside the user directory. This number represents 3.26% of the samples. Actually, this number is surprising low. We would have expected an increase of samples that abstain from tampering with files outside the user directory since the launch of Windows Vista and the introduction of the User Account Control (UAC) in January 2007. The UAC limits the privileges of executables to user privileges. The user would have to explicitly authorize modification of files outside the user directory.

6.14 Harvesting of Email Addresses

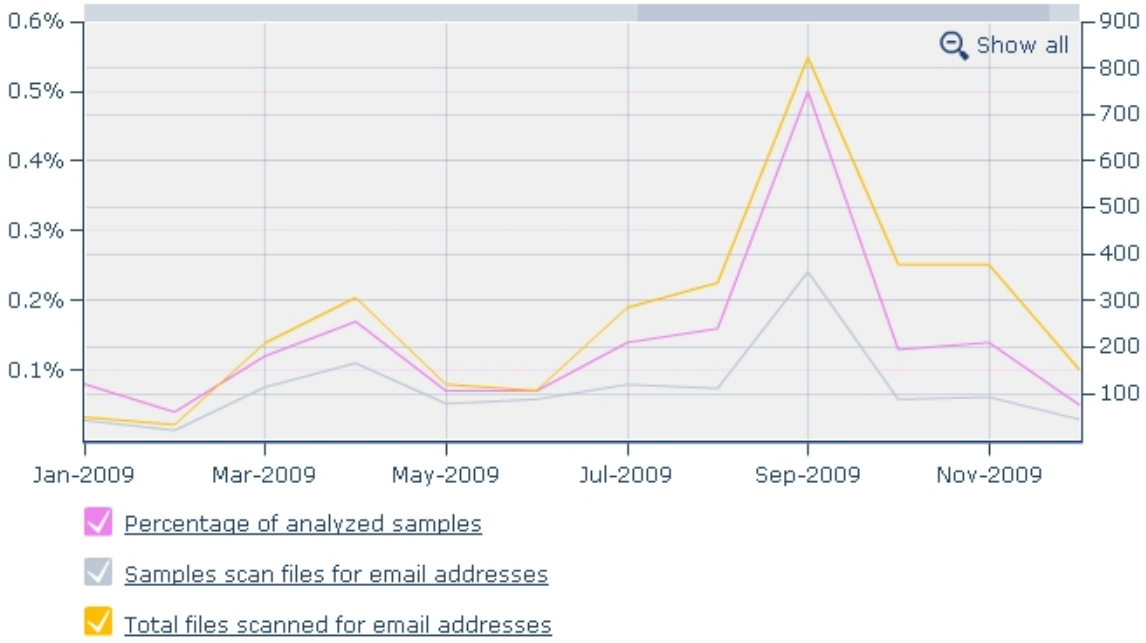


Figure 6.20: Monthly email address harvest

The chart in Figure 6.20 describes samples that harvest email addresses by performing read operations on address books and email files. The samples are grouped according to their analysis end and can be viewed in daily and monthly intervals.

The environment in which we perform the analyses has Microsoft Outlook Express installed. Outlook Express' address book and email files save the email addresses in plain text. It is an easy task for malware to extract these email addresses and report them back home. In Table 6.10, we list the paths of the files that are used by Outlook Express to store email addresses. The first asterisk replaces the user name. Previously we had a user with the user name '`user`'. We replaced this user with the user '`Administrator`' since the previous user name was utilized

by malware to recognize the Anubis environment. The second asterisk stands for the identity ID which is unique for any user. To identify a sample that performs email address harvesting, we simply scan its read operations for one of these files.

File name
C:\Documents and Settings\Administrator\Application Data\Microsoft\Address Book\Administrator.wab
C:\Documents and Settings*\Local Settings\Application Data\Identities*\Microsoft\Outlook Express\Folders.dbx
C:\Documents and Settings*\Local Settings\Application Data\Identities*\Microsoft\Outlook Express\Inbox.dbx
C:\Documents and Settings*\Local Settings\Application Data\Identities*\Microsoft\Outlook Express\Offline.dbx
C:\Documents and Settings*\Local Settings\Application Data\Identities*\Microsoft\Outlook Express\Outbox.dbx
C:\Documents and Settings*\Local Settings\Application Data\Identities*\Microsoft\Outlook Express\Pop3uidl.dbx
C:\Documents and Settings*\Local Settings\Application Data\Identities*\Microsoft\Outlook Express\Sent Items.dbx
C:\Documents and Settings\user\Application Data\Microsoft\Address Book\user.wab
C:\Documents and Settings*\Local Settings\Application Data\Microsoft\Outlook\Outlook.pst

Table 6.10: Outlook Express files that are monitored to detect email address harvesting.

- **Percentage of analyzed samples**: This graph gives the percentage of analyzed samples that perform read operations on one of the files listed in Table 6.10 and is thus considered to harvest email addresses.
- **Samples that scan files for email addresses**: This graph represents the absolute number of samples that are identified as email address harvesters. It is hidden by default.
- **Total files scanned for email addresses**: This graph shows the total number of files that have been scanned for email addresses. Since we monitor only 14 files, 14 is the maximum value here for a single sample. Multiple read operations on the same file would be counted as well, but we have only seen three samples that perform multiple reads on one of the monitored files, so this number can be neglected. This graph is not visible by default as well.

Only 2 000 samples performed a read operation on at least one of the files listed in Table 6.10. In total, Outlook Express' files were read 4 900 times. On average one of these samples scans 2.5 files for email addresses. `Administrator.wab` is the most popular file and has been scanned over 800 times.

6.15 Modification of System Files

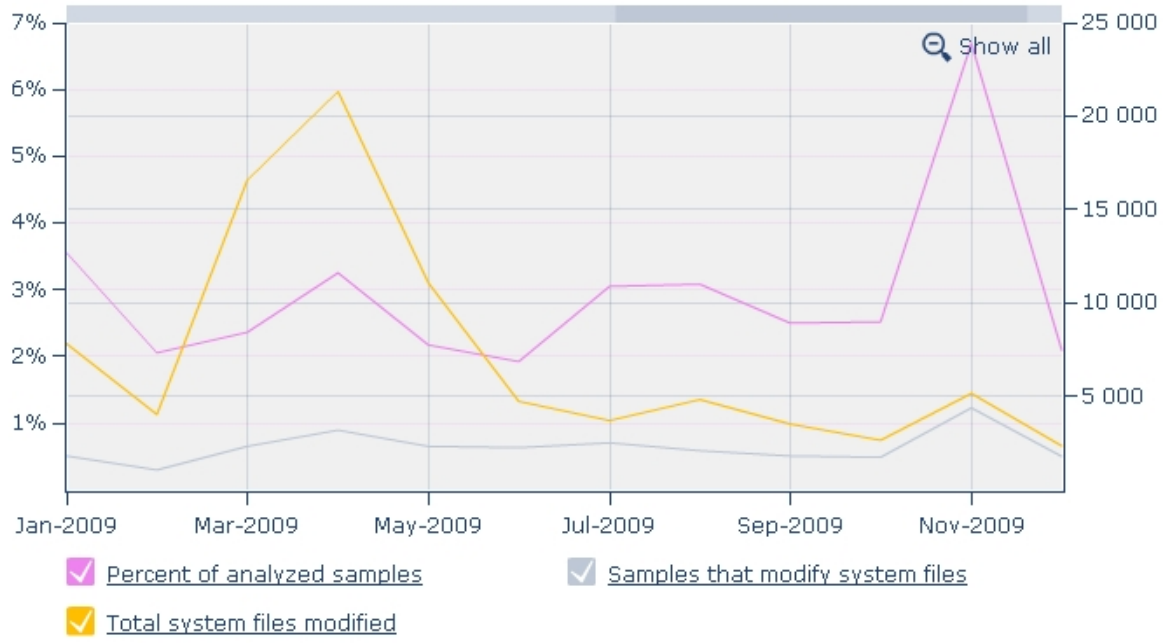


Figure 6.21: Monthly modification of system files

Figure 6.21 gives an overview on samples that modify Windows system files in 2009. The user can switch between a daily and a monthly view on this statistic. The analysis end determines the group affiliation to a date of the chart's y-axis.

Windows system files reside in the directory '`C:\Windows\`'. Although program files like those of Internet Explorer, which are located in '`C:\Program Files\Internet Explorer\`', can be counted as system files, we decided not to monitor these. A sample is considered to modify Windows system files, if it modifies a file residing in one of the directories listed in Table 6.11 or in a subdirectory of these. Since a file creation entails a modification of this file, if it is not intended to create an empty file solely, we have to make sure that samples, listed in this category, do not only change the files that have been created by them. Although a file might be created in one of the monitored directories, it is not considered a system file, because it did not come with the installation of Windows. Hence, we have to check, whether a modified systems file is not created by the sample as well.

Directory name
`%windir%\`
`%systemroot%\`
`C:\Windows\`

Table 6.11: Locations for Windows system files.

In our first attempt we saw many samples that modify Windows event files. Windows event files are located in the directory 'C:\Windows\system32\config\'. Further investigation showed that this modification is performed by services.exe in the course of its normal operation. We include services.exe in our analysis whenever the sample creates a Windows service or interacts with an existing service. Most of the events are logged to SysEvent.evt and AppEvent.evt. Therefore, we decided to exclude modifications of event files.

- **Percentage of analyzed samples**: This graph gives the percentage of new analyzed samples that modify a system file.
- **Samples that modify system files**: This graph represents the number of samples that modify system files. It is not displayed by default.
- **Total system files modified**: This graph sums the number of system files that have been modified and is hidden by default either.

A total of 59 000 samples modified 166 000 system files. Altogether, 3.39% of the samples modify 2.8 system files on average. The peak value in November 2009 stands out. Almost 60% of the samples that modify system files in November 2009 were IRC bots submitted by the user 'hybrid-analysis' to test the effectiveness of Reanimator with real-world datasets[50]. 'C:\Windows\system32\drivers\etc\hosts' is the most popular target for these IRC bots. They commonly update URL-to-IP mappings to prevent access to websites that provide updates for security products. The system files, generally modified most often are examined in the following section.

6.16 Top Modified System Files

The chart presented in Figure 6.22 lists the top ten modified system files in October 2009. The top ten are generated on a monthly basis and can be viewed for each month.

A system file is considered modified if the description in Section 6.15 applies. All modified system files in a month equal 100%. C:\Windows\system32\drivers\etc\hosts is the most popular system file to be modified. It resides in the top ten throughout the observation period. The hosts file holds Windows' mapping for hostnames to IP addresses. It can be used by malware to redirect well known host names like http://gmail.com or a bank's net banking site to a server controlled by the malware's author. Phishing attacks are an easy task with this technique. Another intention of this file modification might be to redirect domain names of antivirus vendors in order to prevent software updates of antivirus products.

Another popular target is C:\Windows\SYSTEM.INI. This file specifies device drivers and the default shell loaded by Windows during boot time. Editing this file allows malware to survive a restart of the system, as it can be executed again at boot time.

Additionally, Windows system executables are a popular target for modification. These files are probably being patched so they execute arbitrary code, once they are run.

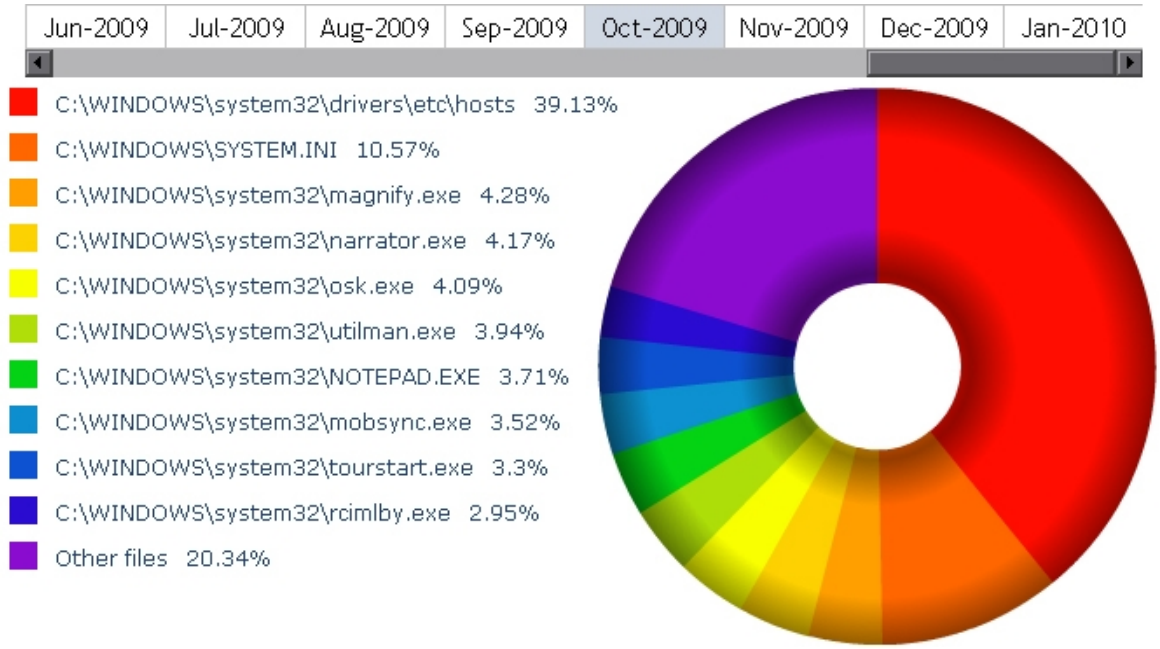


Figure 6.22: Top modified system files in October 2009

6.17 Network Activity

Figure 6.23 gives an overview of the samples that perform some kind of network activity in 2009. This graph is available in both monthly and daily intervals. The samples are grouped by the date of their analysis end.

Network activity includes all types of network communication regardless of the actual protocol used. In fact, a sample could use its own TCP or UDP protocol for communication and this communication would still be recognized as network activity. Even unsuccessful connection attempts are considered as a kind of network activity.

- **Percentage of analyzed samples**: This graph gives the percentage of new analyzed samples that perform some kind of network activity.
- **Samples with network activity**: This graph represents the number of samples that are identified to communicate over the network. This graph is not visible by default.

Altogether, 0.8 million samples perform network activity. This number represents 45.49% of all analyses. The low percentage in July 2009 is noticeable. By inspecting the statistics on popular network protocols, we can conclude that this decline is due to less HTTP activity in that month. The low values for DNS activity for July 2009 is probably a cause of less HTTP activity as well because it is very common to address HTTP server via a DNS name.

It is surprising that less than half of the samples perform network activity. This is probably due to the limited time we observe the behavior of a sample. A key logger, for example, might

Figure 6.23: Monthly network activity

gather information for several hours or days before it sets up a network connection to report its findings. Additionally, some samples do not work as intended in Anubis because they miss components or behave intentionally different because they are capable of detecting that they are being analyzed.

6.18 HTTP Activity

The chart in Figure 6.24 represents the samples whose network traffic contains HTTP connections. Again, this chart can be viewed in monthly and daily intervals. The date of the analysis end is used to determine the sample's affiliation.

The chart's data is not based on the analysis report. The network traffic produced by a sample is stored in a network dump. For the FIRE [14] system, we extract HTTP communication directly from this network dump and save it in the database. To identify a rogue network, we require the data on HTTP communication in more detail than it is available in the analysis report. This is why we do not extract the information on HTTP conversations from the report but reuse the data, already available in the database.

- **Percentage of analyzed samples**: This graph represents the percentage of samples that perform HTTP activity.

- **Samples with HTTP activity**: This graph gives the number of samples that use HTTP to communicate with other hosts. It is hidden by default.

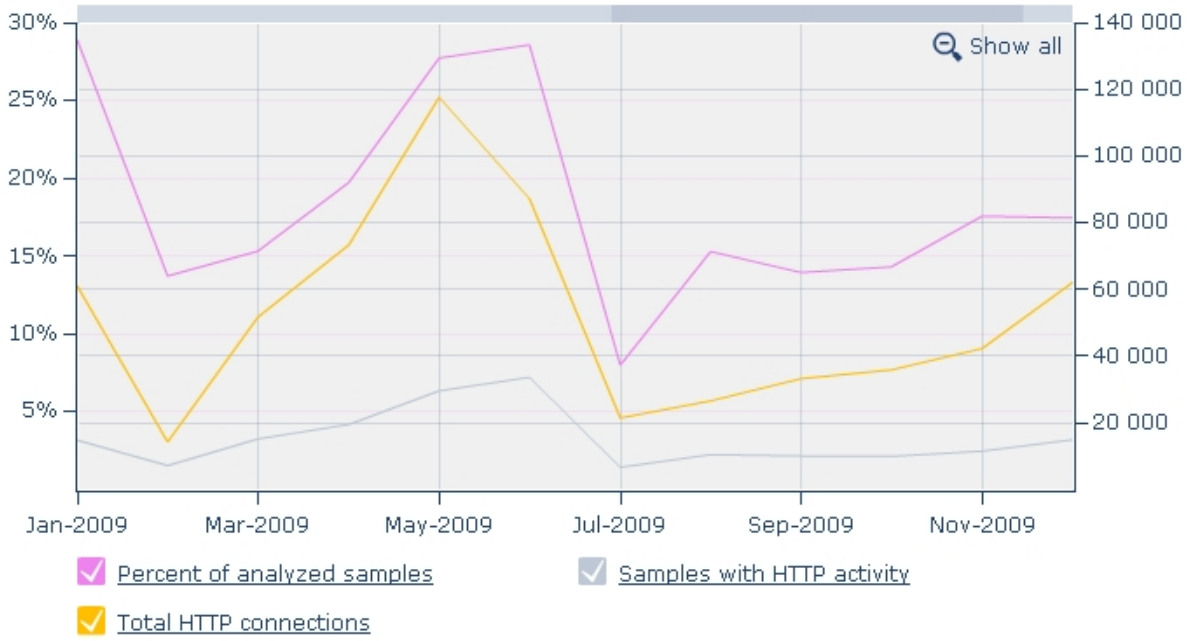


Figure 6.24: Monthly HTTP activity

- **Total HTTP connections**: Here we count the number of HTTP connections established by the samples. Each `get`, `post` or `head` request increases the number of HTTP connections by one. By default, this graph is not visible.

1.2 million HTTP connections were established by 0.3 million samples. On average, each sample sends four requests to a HTTP server.

We distinguish three different HTTP request methods. The `get` method is by far most used and has a share of 91.15% of the HTTP requests. In contrast, only 7.36% perform a `post` request and 1.49% send a `head` request.

It is not surprising that over 40% of the samples with network activity actually use HTTP to fulfill this task. HTTP is the most widespread protocol on the internet. Therefore, it is very unlikely that HTTP is blocked by some firewall, thus allowing a reliable form of communication. It is often used to report successful infection of the host or to download additional instructions or malware. Via a `post` request, it is possible to upload files to a server, possibly under control of the malware author. These files could contain login credentials or credit card numbers collected on the infected machine.

6.19 HTTP Server

The chart in Figure 6.25 lists the top ten HTTP servers for November 2009. The top ten are calculated for each month.

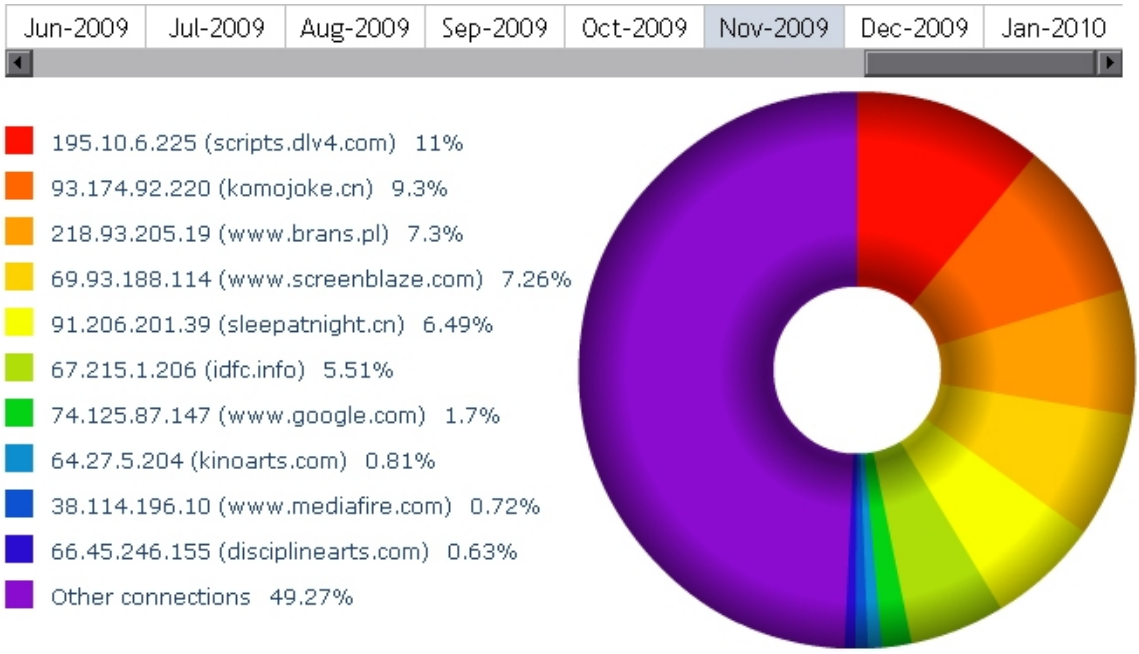


Figure 6.25: Top HTTP server in November 2009

It stands out that the hosts are very diverse. No host appears in the top ten for more than 5 months. Most of the older hosts are no longer reachable. But investigation of the server response shows that most of the servers send binaries as a result of the malware request. Usually, they host less than five different binaries.

Recently, scripts.dlv4.com is very popular. This site is disguised as a hardcore porn site but it hosts malware which is downloaded during analysis. We have identified 13 687 downloads of two different binaries from this host.

More than 13 000 times, a HTTP request has been sent to www.google.com. Looking through the search queries, we found that arbitrary strings like '`irrnaptdkx`' or '`xfgnnefbwv`' were often used as search terms. Although, none of these search terms returned any results at the time of our manual search, we suspect that malware authors try to optimize a web page for one of these key words. The sample would not have to know the URL of the server but only the search term and could use Google to retrieve the required URL. This would be an effective technique if the URL of contact server is subject to frequent change. Although, we found only one request to a site, cached by Google, the desired information may be extracted from Google's cache, if the URL is not reachable at all.

6.20 SMTP Activity

The chart presented in Figure 6.26 lists the samples whose network traffic contains SMTP connections in 2009. It can be viewed in monthly and daily intervals. The sample's analysis end is used to group the samples to their time periods.

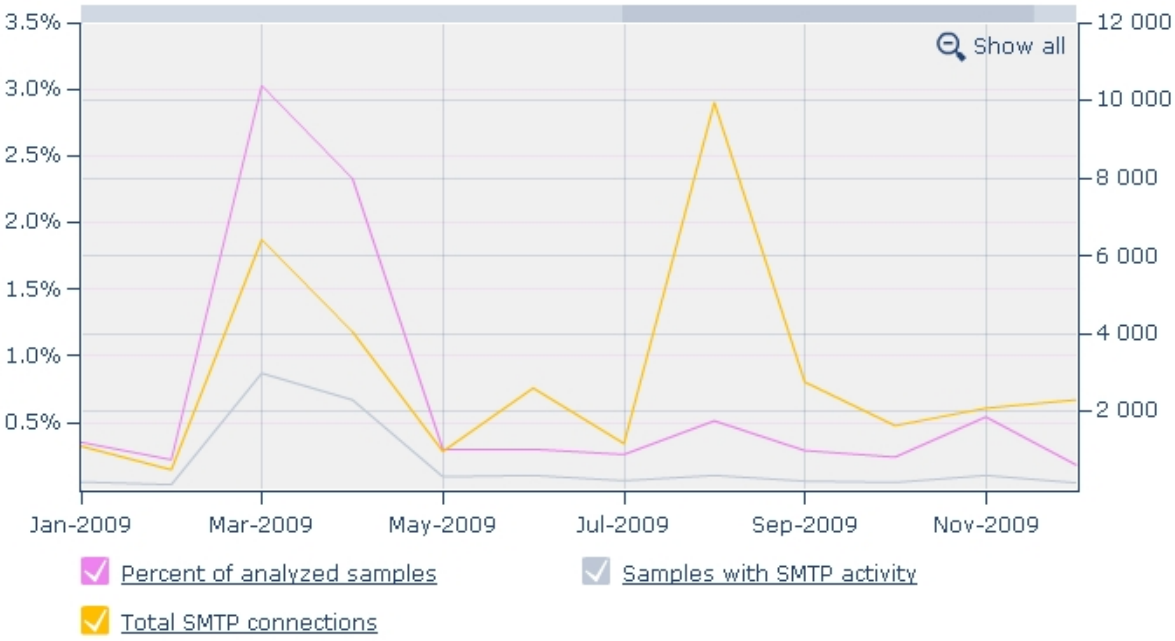


Figure 6.26: Monthly SMTP activity

SMTP is used to transfer emails. Emails sent by malware often contain unwanted advertisements or they attach themselves or other malware to emails in order to propagate malware. But emails sent during the analysis in Anubis do not actually reach the recipient. Instead, all traffic targeting a SMTP server is redirected to our SMTP server that simulates the protocol but does not forward the emails.

- **Percentage of analyzed samples**: This graph gives the percentage of analyzed samples that perform SMTP activity.
- **Samples with SMTP activity**: This graph represents the number of samples with SMTP activity. This graph is not visible by default.
- **Total SMTP connections**: This graph shows the number of emails that have been sent in each time period. For a sample that sends emails it is very common to send more than one. By default, this graph is not displayed.

A total of 13 000 samples have been identified to transmit emails. This number represents 0.76% of all samples analyzed. Altogether, almost 57 000 SMTP connections were detected. On average each sample sent 4.3 emails.

The number of samples sending spam is surprisingly low. This is probably due to the limited time we run samples.

It stands out that more than 3% of the samples analyzed in March 2009 send 6 400 emails. Most of these emails seem to inform the author of the malware of a successful infection and send

details on the infected operating system and the infected host's IP address. 10% of the mails sent in this month have a binary attached. On the basis of these mails' subject, these mails try to trick the recipient to open the attachment by pretending to be an undelivered email or some information related to the recipient's email account.

In August 2009 we do not have an unusual high number of samples that send emails but the samples send 28.5 emails on average. Taking a closer look at these emails, we found 7 000 emails that have hexadecimal code in the subject and the body. All of these emails have Flash files attached to it. It looks to us as if these emails use an erroneous encoding and thus cannot be decoded properly by our protocol analysis. Since most of these samples contact an Ukrainian IRC server, we suspect it to be some Cyrillic character encoding.

Interestingly, 7.3% of the emails are sent to '`email[01-10]@[domain]`'. We doubt that these addresses actually exist but suspect that these emails are sent to test the connectivity to the SMTP server.

6.21 SMTP Server

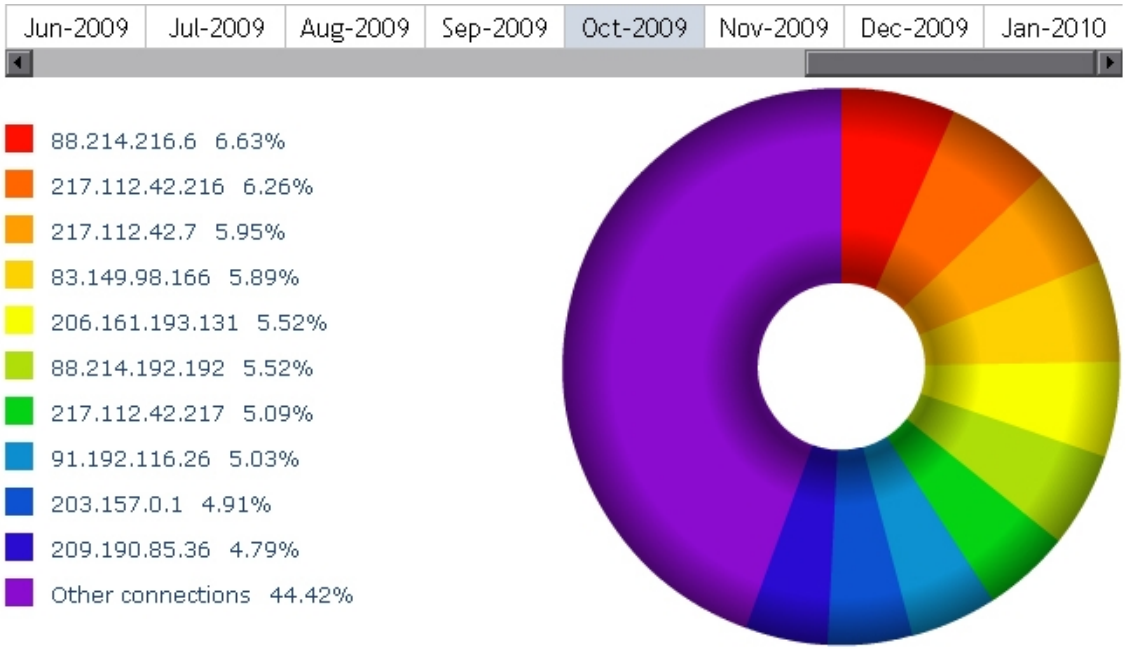


Figure 6.27: Top SMTP server in October 2009

The chart in Figure 6.27 lists the top ten SMTP servers for November 2009. The top ten are calculated on a monthly basis.

It is noticeable that the SMTP servers contacted are very diverse. The top ten rarely exceed 50% of all servers contacted. In Table 6.12, we list the top ten owners of the SMTP servers that were contacted by our samples. We used the companies' IP ranges to calculate this ranking. All

SMTP server's owner	Percentage
Microsoft Corp	9.55%
Google Inc.	6.50%
GMX GmbH	4.39%
Valuehost	3.12%
Hosting Solutions Ltd.	2.15%
TU Wien	3.13%
Mail.Ru	1.97%
Vortech Inc.	1.81%
Yahoo! Inc.	1.70%
LeaseWeb	1.10%

Table 6.12: Owners of the most contacted SMTP servers.

of these companies offer free webmail accounts. In Section 6.20, we have ascertained that a considerable amount of emails is sent to inform the malware's author of a successful infection. It is understandable that the malware authors do not use an email address that is easily traceable to their real identity for this task but create an anonymous email account at a free email provider.

Although seclab.tuwien.ac.at does not offer a free webmail service, 1% of the samples contact this domain's SMTP server. Moreover, 3.13% use a SMTP server residing in the IP range of the University of Technology Vienna. The malware probably identifies the domain it is currently executed in and tests the connectivity to its SMTP server.

6.22 IRC Activity

The chart represented in Figure 6.28 describes the samples that perform IRC activity for 2009. It is available in monthly and daily intervals. The sample's analysis end determines its grouping.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that are identified to perform IRC activity.
- **Samples with IRC activity**: This graph shows the number of samples with IRC activity. This graph is not visible by default.
- **Total IRC connections**: This graph lists the total IRC connections for the samples in a time period. By default, this graph is not shown.

In total, 24 000 samples performed IRC activity. That represents 1.41% of all analyzed samples. Altogether, 26 000 IRC connections were detected.

87.75% of the samples that use IRC establish only one IRC session. A maximum of five IRC sessions for a single sample were detected. This means that the number of different IRC connections for a sample usually is very low. Only 16.83% of the IRC servers require a password. Generally, the remaining servers do not use very sophisticated passwords. On average, each

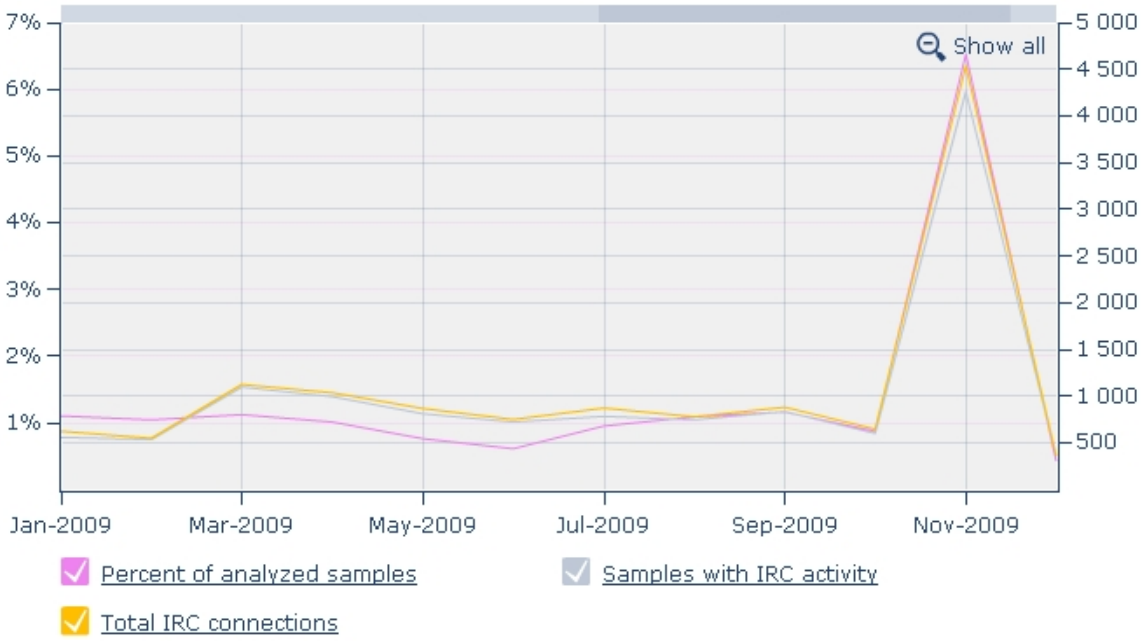


Figure 6.28: Monthly IRC activity

sample joins two channels. 49.13% of the channels are secured by a password. Surprisingly, 94% of the unsecured channels reside on an unsecured server. These channels do not have any access control at all. Interestingly, the sample idles in two out of three channels but the messages sent to channels usually contain status updates and confirmations of completed tasks. Private messages, on the other hand, mainly contain commands to be executed by the samples. We recorded 6 000 private messages to a user.

The peak value in November 2009 is due to experiments that we performed in order to test the effectiveness of a new prototype, Reanimator [50]. In this experiment, we re-submitted and re-analyzed 14 000 samples that showed IRC activity in their last analysis. But only 4 200 samples could connect to their IRC server in this analysis. The IRC servers of the other 10 000 samples are probably not reachable anymore and thus Anubis could only identify a connection attempt but no IRC connection.

6.23 IRC Server

Figure 6.29 lists the top ten IRC server in October 2009. The top ten can be viewed for each month.

Recently, most IRC connections are established with servers residing in the IP range of the Dutch ISP ‘XS4ALL’. Generally, servers of the Canadian web hosting provider ‘GloboTech Communications’ seem most popular for the hosting of IRC servers.

Table 6.13 lists the port numbers that the IRC servers listen on. In the second column, we list the protocols that are registered to the ports [20]. It stands out that only 21.44% of the IRC

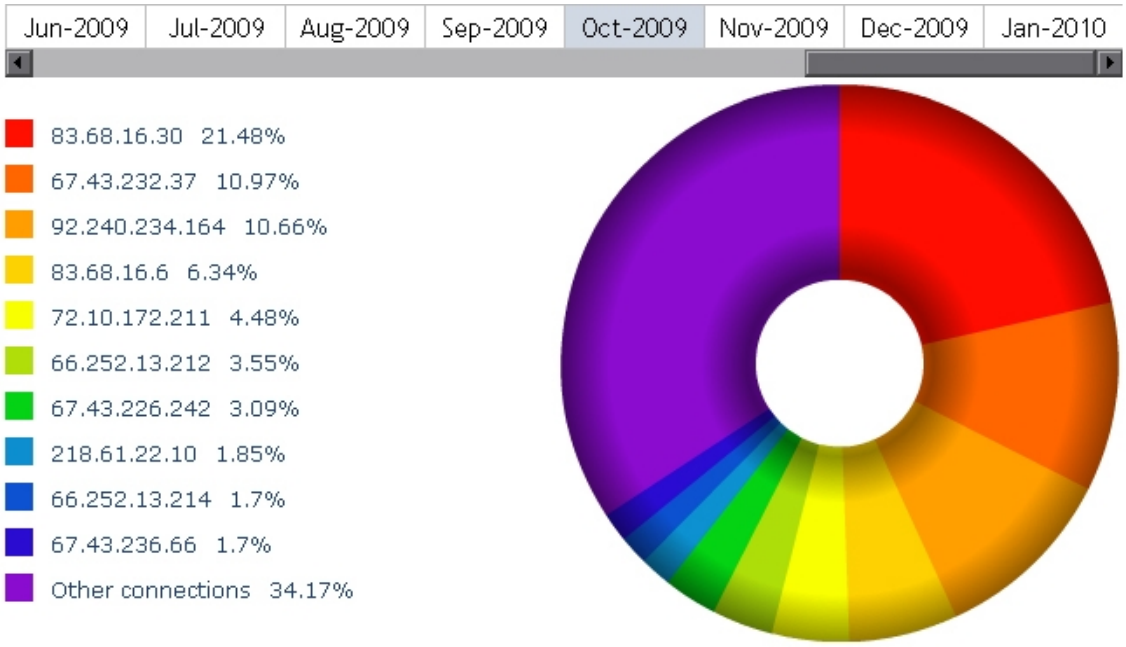


Figure 6.29: Top SMTP server in October 2009

Port	Standard for	Percentage
6667	IRC	21.44%
80	HTTP	10.72%
8080	HTTP	8.46%
10324	Unassigned	5.34%
1863	MSNP	5.24%
5190	AOL	4.53%
7000	AFS3-FileServer	4.14%
3305	Odette-FTP	2.43%
13001	Unassigned	2.28%
1867	UDrive	2.03%

Table 6.13: Most popular ports used by IRC servers.

servers actually use the standard IRC port. This is probably due to the fact that many desktop firewall products, like ZoneAlarm, block all outgoing connections to port 6667 by default. To bypass firewalls, most IRC servers listen on ports assigned to other well known protocols like HTTP or ports used by popular applications like Windows Live Messenger or ICQ. These ports are commonly not blocked because they are frequently required by the user for legitimate tasks.

6.24 FTP Activity

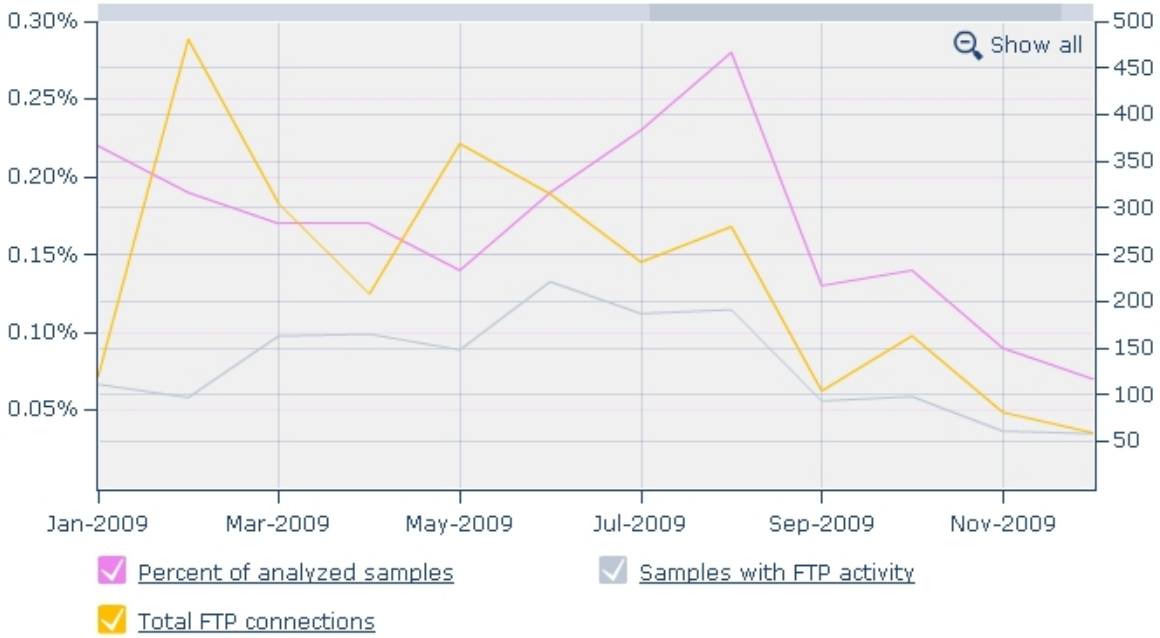


Figure 6.30: Monthly FTP activity

The chart illustrated in Figure 6.30 describes samples whose network traffic contains FTP connections in 2009. The sample's analysis end determines the grouping. This chart is available in daily and monthly intervals.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that perform FTP activity.
- **Samples with FTP activity**: This graph gives the absolute number of samples with FTP activity. This graph is not visible by default.
- **Total FTP connections**: This graph shows the number of FTP connections. It is hidden by default.

Altogether 3 000 samples were detected that perform FTP activity. This number represents 0.18% of the samples. A total of 6 400 FTP connections are recorded. On average each sample connects to 2.1 FTP servers.

99.39% of the servers contacted actually listen on port 21. This port is the standard port for FTP communication. Although the user name has been accepted by the server in 99.06%, only 46.4% of the login attempts are successful. Even though we recorded a high number of unsuccessful login attempts, we did not see any brute force attacks to gain access to FTP servers. But we found few samples that repeated an unsuccessful login attempt several hundred times with the same login credentials.

6.25 DNS Activity

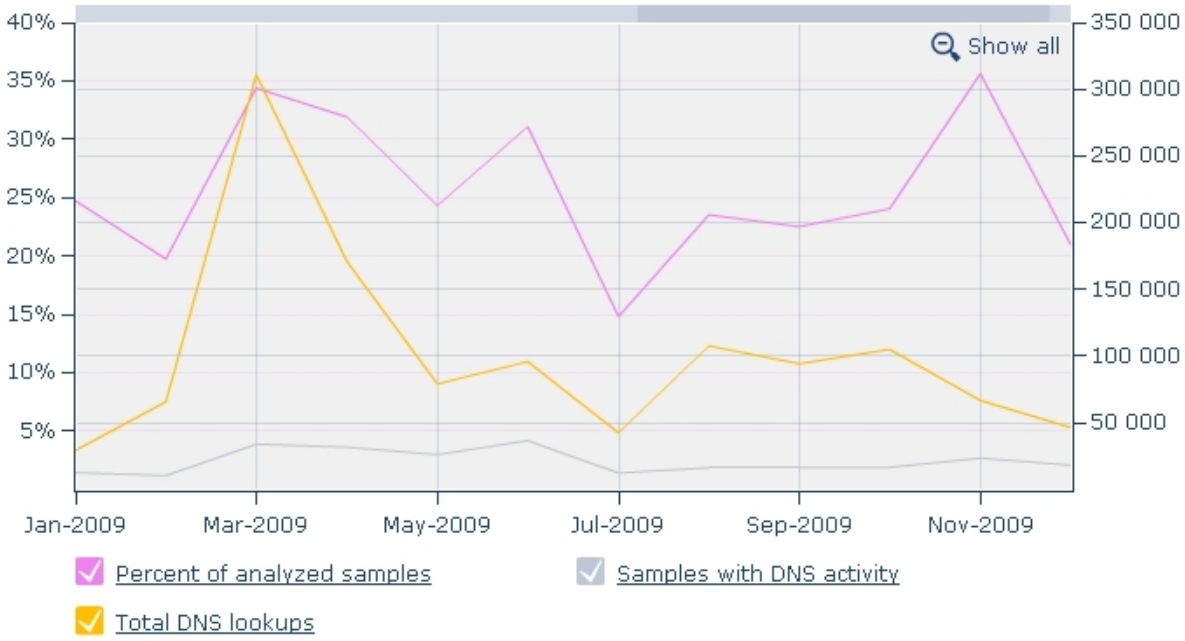


Figure 6.31: Monthly DNS activity

The chart presented in Figure 6.31 describes the samples that perform DNS lookups in 2009. Again, the sample's analysis end is used to group the samples. This chart is available in monthly and daily intervals.

- **Percentage of analyzed samples**: This graph gives the percentage of analyzed samples that perform at least one DNS lookup.
- **Samples with DNS activity**: This graph represents the number of samples that perform a DNS lookup. This graph is hidden by default.
- **Total DNS lookups**: This graph shows the total number of DNS queries that have been detected in the respective time frame. By default, this graph is not visible.

In total, 25.29% of the samples query a DNS server. That are 0.4 million samples performing 2 million lookups. On average, each sample completes 5 DNS queries.

It is not surprising that an overwhelming majority of the samples with DNS activity, 99.72%, perform a lookup on an IPv4 address record. 0.13% query a DNS server for a mail exchange record and 0.13% perform a reverse DNS lookup. Most of the DNS queries are successful, but a significant percentage of 9.44% fail. Most of the DNS lookups that fail are due to a DNS query to 'wpad'. With this query, samples probably try to use the Web Proxy Auto-Discovery Protocol (WPAD) to download the proxy auto-configuration file for the network they are currently in [38].

In Anubis this query fails because no WPAD servers are configured in its network. The remaining 4.30% of unsuccessful DNS query might occur because the according servers have been taken offline already, most likely by the malware author himself in order to evade detection and to make it more difficult to track and stop him [26].

Query	Percentage
wpad	5.14%
d.trymedia.com	2.84%
www.fenomen-games.com	1.79%
www.gamecentersolution.com	1.24%
scripts.dlv4.com	1.23%
proxim.ntkrnlpa.info	1.01%
www.screenblaze.com	0.97%
proxim.ircgalaxy.pl	0.95%
amlocalhost.macrovision.com	0.88%
www.google.com	0.66%

Table 6.14: Most queried DNS names.

In Table 6.14, we list the top ten DNS queries. Most of the top queried domains can be found in the top ten of HTTP connections as well. HTTP is clearly the main cause for our samples to perform DNS queries.

6.26 Address Scans

The chart illustrated in Figure 6.32 represents the samples that perform address scans in 2009. This chart is available in monthly and daily intervals. The analysis end determines the grouping of the samples.

An address scan is detected by Anubis if a sample contacts more than ten addresses on the same port. Because HTTP's standard port 80 and SMTP's standard port 25 are used very frequently for tasks that are other than scanning addresses, we excluded addresses that are contacted on either of these ports.

- **Percentage of analyzed samples**: This graph represents the percentage of the analyzed samples that perform address scans.
- **Samples that perform address scans**: This graph shows the number of samples that scan IP addresses.
- **Total address scans**: This graph gives the total number of the performed address scans.

A total of 250 000 samples, which represents 14.02% of all samples, performed 520 000 address scans. On average 2.08 address scans are performed by each of these samples.

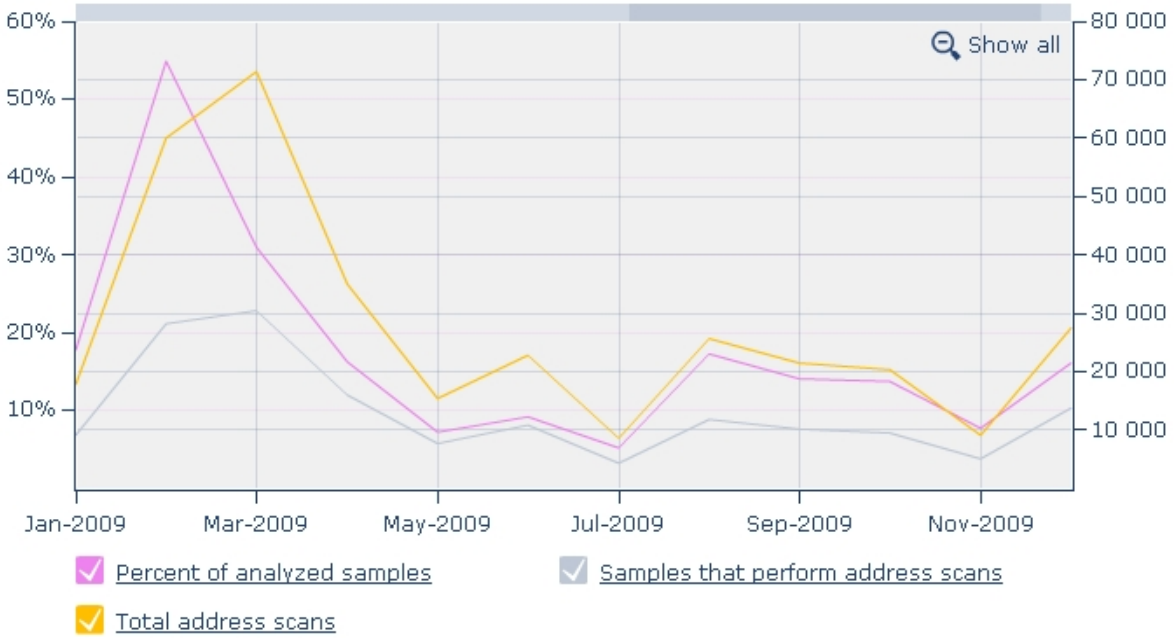


Figure 6.32: Monthly address scans

Almost all the scans are performed via TCP. Only 0.18% utilize UDP. 87.42% try to find computers that are listening on NetBIOS ports 445 and 139. NetBIOS is used for file sharing between Windows hosts and is prone to vulnerabilities [40, 39]. These samples are probably NetBIOS worms that exploit Windows' NetBIOS vulnerabilities to propagate. Another 10% scan addresses on port 9988 for Radmin installations with weak passwords [36]. Radmin is a remote-control software by Famatech. Since Radmin listens on port 4899 by default [9], we think that the Radmin installations listening on port 9988 were installed as backdoors by other malware.

Remarkably, the number of address scans is significantly higher in February and March 2009. The distribution of remote ports that were used for the scans does not differ from the remaining address scans. But the target subnets for the scans are more diverse.

Subnet	**Percentage**
128.130.0.0/16	6.90%
no_subnet	6.70%
192.168.0.0/16	1.76%
128.130.56.0/24	0.59%
192.0.0.0/8	0.23%

Table 6.15: Most scanned subnets.

In Table 6.15, we list the most scanned subnets and the percentage of the scanned addresses belonging to these subnets. It stands out that most scans reside in the subnet of the infected host.

This is not surprising as malware authors hope to find vulnerable hosts within the same network and thus within a trusted zone, possibly not protected by individual firewalls. However, not all scanned IP addresses lie in a simple subnet. We also see that an almost equal amount of addresses seem to be randomly chosen.

6.27 Size of Downloaded Files

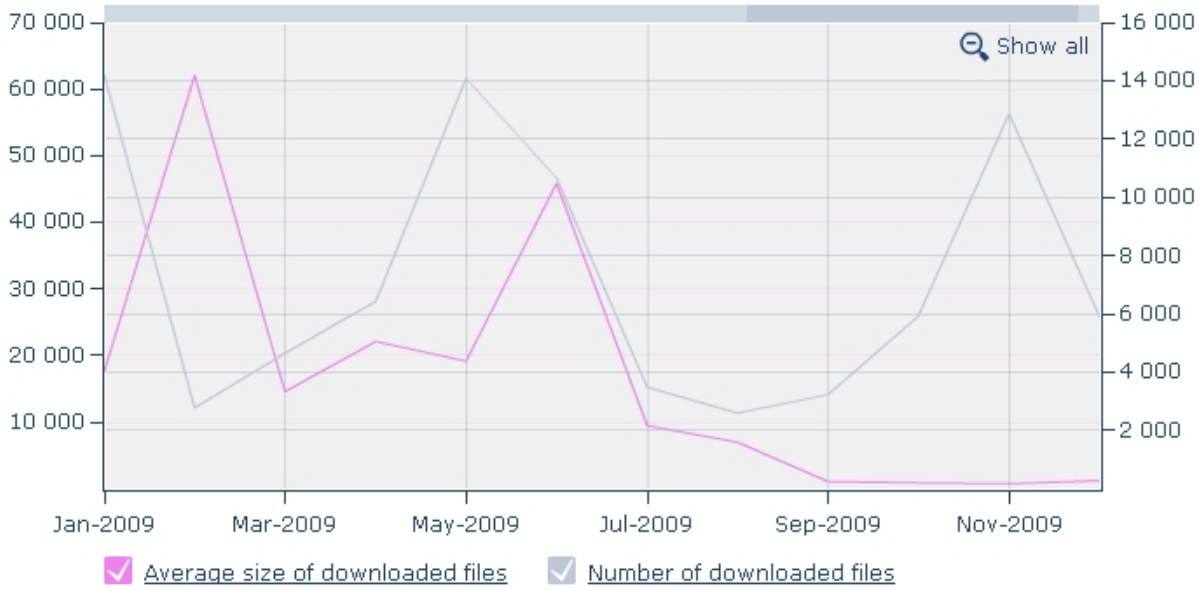


Figure 6.33: Monthly average size of downloaded files

The chart in Figure 6.33 shows the average size of executables that were downloaded during the analysis in 2009. This chart can be viewed in monthly and daily intervals. The end of a sample's analysis specifies its affiliation.

We scan HTTP requests for a response that has the content type set to `application/octet-stream*` to identify downloads of executables. To determine the size of an executable, we rely on the `content length` field of the server's response. This approach is by no means perfect since a misconfigured web server could return a wrong content type or a wrong content size for a file. Nevertheless, this method allows us to get a good estimate of the actual number of downloaded files.

- **Average size of downloaded files**: This graph represents the average size of downloaded executables in kilo byte.
- **Number of downloaded files**: This graph represents the number of executables that were downloaded in the respective time period. Although this graph uses a different scale, which is plotted onto the right y-axis, than the first, we still display it by default.

Altogether, 200 000 executables with an average size of 14.6 MB were requested via HTTP.

Unfortunately, 1.28% of the server responses for executables return a content length of more than 100 MB. Probably, some of these responses contain an erroneous content length but still cause a significant increase of the average size of downloaded executables in February and June 2009. Generally, this chart only gives an upper bound for the average size of second stage malware. But we can see that the actual size of second stage malware is likely to be well below 1 MB.

6.28 Registry Value Modification

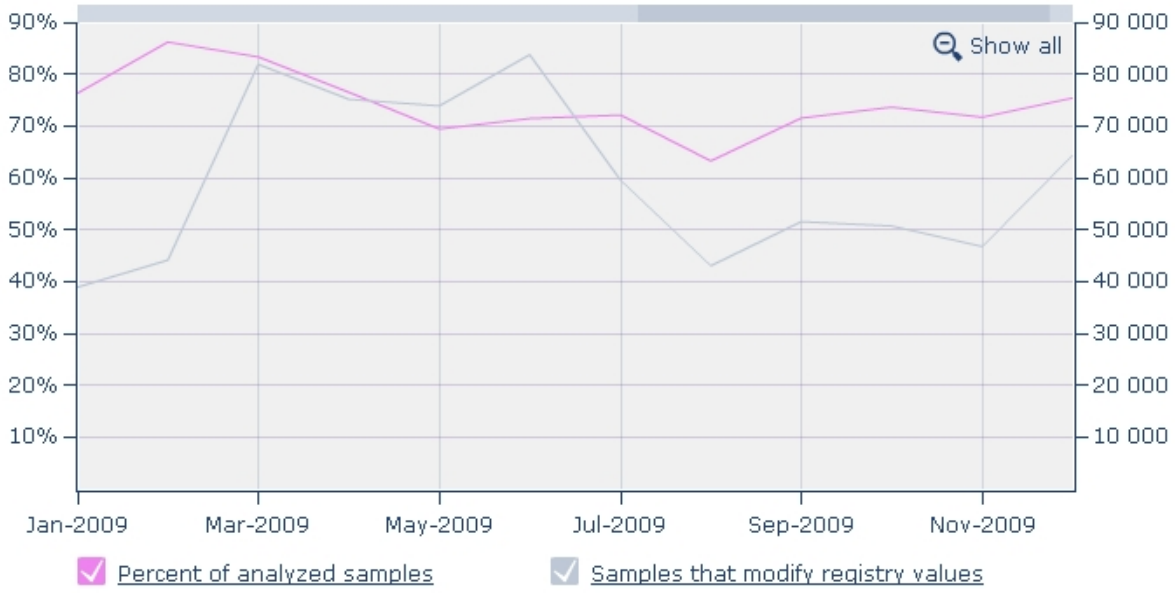


Figure 6.34: Monthly modifiaction of registry values

The chart in Figure 6.34 represents the samples that modify registry values in 2009. The analysis end determines the grouping of the samples. This chart is available in monthly and daily intervals.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that modify registry values.
- **Samples that modify registry values**: This graph shows the number of samples that modify registry values. This graph is hidden by default.

Altogether, 1.3 million samples modified at least one registry value. This represents 73.65% of the samples. On average, each sample modifies 24.2 registry values.

The actual values modified are very diverse. Since we randomize the user id, which is part of the registry key for all keys in the `HKEY_USERS` hive, identical modifications of values in

this branch cannot be identified easily. Nevertheless, we found that 2.15% of the modifications target the registry value `HKLM\Software\Classes\CLSID\{35349B95-82D3-1178-19ED-0E5D2312F5C0}` and another 2.15% modify the value `HKLM\Software\Classes\CLSID\{35349B95-82D3-1178-19ED-0E5D2312F5C0}\LocalServer32`.

The first value contains a random string of 16 characters and the second value holds a path to `C:\WINDOWS\system32\urdvxc.exe` which is characteristic for the Allaple worm. 171 000 samples perform this kind of registry value modification. Thus, we can conclude that Allaple consumed 10% of our analysis time.

6.29 Registry Key Creation

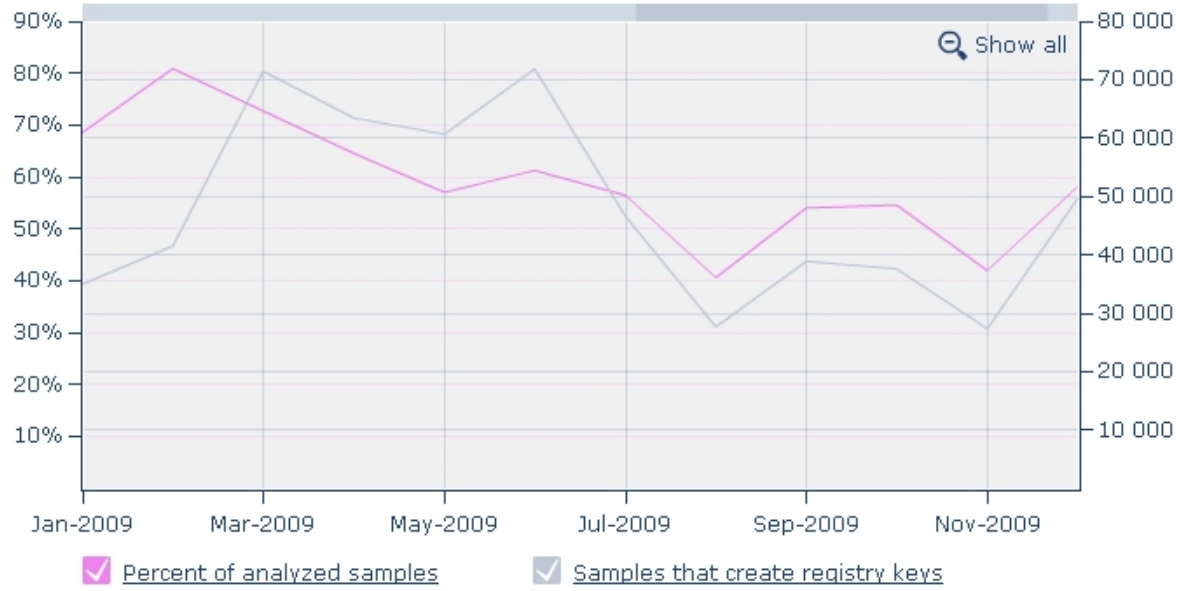


Figure 6.35: Monthly creation of registry keys

The chart illustrated in Figure 6.35 shows the samples that create new registry keys in 2009. The sample's analysis end determines its grouping. This chart can be viewed in daily and monthly intervals.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that create new registry keys.
- **Samples that create registry keys**: This graph shows the number of samples that create at least one registry key. This graph is hidden by default.

In total, Anubis analyzed 1 million samples that create new registry keys. This number represents 58.79% of all samples analyzed. On average, each of these samples creates 9.1 new registry keys.

It is not surprising that the number newly created registry keys is significantly less than the number of modified registry values as a registry value is usually modified in a new key. If a value is located in a key that exists already, it does not have to be created.

6.30 Autostart Capabilities

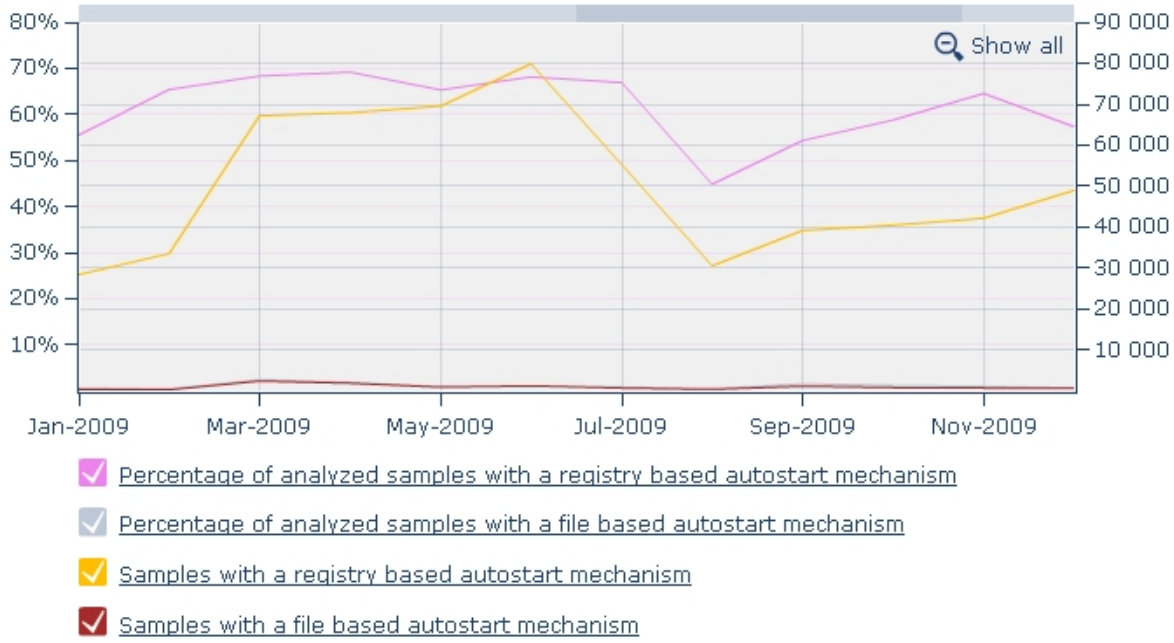


Figure 6.36: Monthly autostart capabilities

The chart in Figure 6.36 shows the samples that are capably of surviving a system reboot by instructing the compromised system to run them at startup. This chart is available in monthly and daily intervals. A samples analysis end determines its grouping.

In this chart, we distinguish two different kinds of autostart mechanisms, file based and registry based autostart mechanism. For the file based autostart mechanism we monitor modifications of the files or content of directories listed in Table 6.17. For the registry based autostart mechanism we observe modifications of registry values in subkeys like `HKLM\Software\Microsoft\Windows\CurrentVersion\Run`. For the complete list of registry keys that we monitor for this statistic refer to Table B.1 in Appendix B.

- **Percentage of analyzed samples with a registry based autostart mechanism**: This graph represents the percentage of analyzed samples with a registry based autostart mechanism.

- **Percentage of analyzed samples with a file based autostart mechanism**: This graph gives the percentage of analyzed samples that modify at least one of the files listed in Table 6.17.
- **Samples with a registry based autostart mechanism**: This graph shows the number of samples that modify at least one registry value in one of the registry keys listed in Table B.1. This graph is hidden by default.
- **Samples with a file based autostart mechanism**: This graph represents the number of samples with a file based autostart mechanism. This graph is not visible by default.

In total, Anubis analyzed 1.088 (55.50%) million samples that utilize a registry based autostart mechanism. Only 20 000 (0.98%) samples use a file based autostart mechanism. Using the registry to add autostart capabilities to a sample is clearly preferred. We think that the reason for the preferential treatment of the registry is that common users are not as familiar with it [42]. The concept of files and the file system are better understood and the users can undo the changes to the file system that lead to automatic execution at startup more easily, especially at locations like the `Startup` folder where a user might stumble upon the malware by chance.

Autostart location	**Percentage**
`HKLM\Software\Microsoft\Windows\CurrentVersion\ Explorer\MountPoints2`	45.39%
`HKLM\System\CurrentControlSet\Services`	21.24%
`HKLM\Software\Microsoft\Windows\CurrentVersion\Run`	15.84%
`HKCU\Software\Microsoft\Windows\CurrentVersion\Run`	11.04%
`HKLM\System\CurrentControlSet\Control\ Session Manager`	7.00%
`HKLM\Software\Microsoft\Active Setup\ Installed Components`	5.04%
`HKLM\Software\Microsoft\Windows NT\CurrentVersion\ Winlogon`	3.97%
`HKLM\Software\Classes\%\shell\%\Command`	2.70%
`HKCU\Software\Microsoft\Windows NT\CurrentVersion\ Winlogon`	2.55%
`HKLM\Software\Microsoft\Windows\CurrentVersion\ RunServices`	1.97%

Table 6.16: Registry based autostart keys.

In Table 6.16, we list the most popular registry keys, which hold values that are modified by malware in order to be executed repeatedly after system reboot. Even though the MountPoints2 key is not documented well by Microsoft, it is the most used registry key to perform this task. This registry key holds information about auto play options of devices that have been mounted [2]. Malware that modifies these registry values accordingly gets executed every time a device is

mounted. Only 23.19% of the samples that modify registry values to persist themselves actually make use of the `Run` key in either the `HKCU` or the `HKLM` hive - 3.69% actually make use of both. These registry keys are used by most benign applications that run at startup, like Skype or MSN.

Interestingly, even though the `RunServices` key is no longer supported by Windows XP [13, 3], 1.97% of the samples with registry based autostart capabilities still make use of this key.

Autostart location	**Percentage**
`C:\Windows\system.ini`	39.84%
`C:\Documents and Settings\*\Start Menu\Programs\Startup\*`	36.23%
`C:\Windows\Tasks\*`	15.93%
`C:\Windows\win.ini`	8.11%
`C:\autoexec.bat`	2.05%
`C:\Windows\winstart.bat`	1.22%
`C:\Windows\wininit.ini`	0.46%
`C:\config.sys`	0.08%
`C:\Windows\System32\autoexec.nt`	0.02%
`C:\Windows\System32\config.nt`	0.01%
`C:\Windows\dosstart.bat`	0.00%

Table 6.17: File based autostart locations.

In Table 6.17, we show the most popular file-based autostart mechanisms. Interestingly, more than a third of these samples creates on average 1.2 entries in the `Startup` folder of the Windows start menu, even though this location is exposed to the user and the entries could be discovered by chance. We did not see any sample that modified `C:\Windows\dosstart.bat`. This not too surprising since this batch file was only used until Windows 98 and was executed if Windows was rebooted into DOS mode [17]. For the sake of completeness, we still monitor this file to detect file based autostart behavior.

6.31 Modification of the Windows Firewall Settings

Figure 6.37 describes the samples that change the settings of Windows Firewall in 2009. It is available in monthly and daily intervals. The analysis end determines the sample's grouping.

The Windows Firewall allows all outbound traffic and blocks all inbound traffic that is not initiated by the user [43, 63]. To open a listening port and to allow inbound traffic to that port, an application has to add itself to a list of authorized applications. During the run time of this application inbound traffic to the specified ports will be allowed.

The settings of the Windows Firewall are stored in the registry. The settings can be set individually for each Windows user account and are applied through various policies. Therefore, we monitor modification of registry values in multiple subkeys. These subkeys are listed in Table 6.18. In this table, we use the alias `HKEY_CURRENT_USER (HKCU)` for the hive

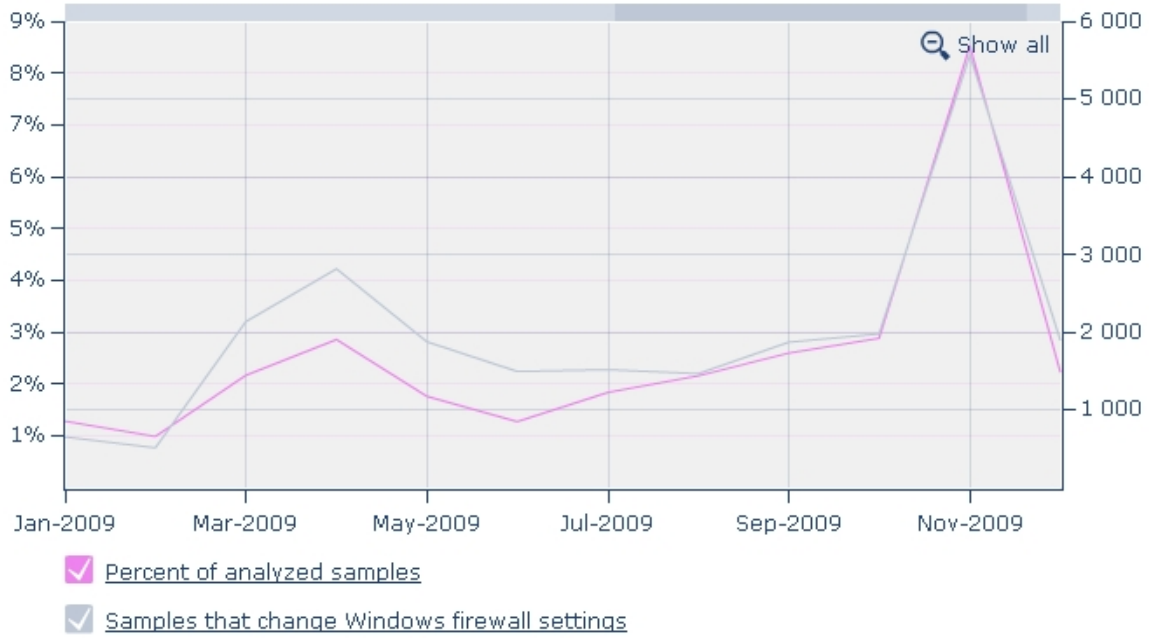


Figure 6.37: Monthly modification of Windows Firewall settings

HKEY_USERS (HKU) with its corresponding user id although the registry keys are not saved this way in the analysis report.

Registry subkey
HKCU\Software\Policies\Microsoft\WindowsFirewall
HKLM\Software\Policies\Microsoft\WindowsFirewall
HKLM\Software\Policies\Microsoft\FirewallPolicy
HKCU\System\ControlSet00*\Services\SharedAccess\Parameters\FirewallPolicy
HKLM\System\ControlSet00*\Services\SharedAccess\Parameters\FirewallPolicy
HKCU\System\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy
HKLM\System\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy

Table 6.18: Subkeys that are monitored for value modification containing Windows Firewall settings.

- **Percentage of analyzed samples**: This graph gives the percentage of analyzed samples that change Windows Firewall settings by altering at least one value of a subkey in Table 6.18.

- **Samples that change Windows firewall settings**: This graph represents the number of samples that change the settings of Windows Firewall. This graph is not visible by default.

Altogether, 42 000 samples change the Windows Firewall settings. This number represents 2.38% of the analyzed samples.

The peak value in November 2009 is due to IRC bots submitted to verify Reanimator [50]. 65.49% of the samples that change firewall settings in this month originate from this set of samples. Although the plain IRC protocol would not need any alterations of the default firewall settings to perform correctly, Client-to-Client Protocols like 'Direct Client-to-Client (DCC)' require listening ports to accept inbound traffic [73].

91.13% of these samples alter the list of applications that are authorized to accept inbound traffic. Interestingly, most of the samples enter names of common Windows system files like `winlogon.exe` or `spoolvs.exe` that probably have been modified previously and will most likely raise less suspicion by the user. Only 13.13% disable the Windows Firewall completely. Disabling the Windows Firewall usually causes an alert by the Security Center. This is probably the reason why malware prefers the first approach.

4.79% of these samples apply both methods to allow inbound traffic even if the firewall is activated again manually.

6.32 Disabling the Windows Update mechanism

Figure 6.38: Monthly disabling of Windows Update

Figure 6.38 describes the samples that disable automatic Windows Update in 2009. The date of the sample's analysis end is used to group them. This chart is available in daily and monthly intervals.

Windows Automatic Update is a service that keeps the system up to date by downloading and installing software updates provided by Microsoft. The settings for Windows Update are stored in the registry at different locations depending on whether they are set for the whole machine, for each user account individually or via policies. In this statistic, we are interested if a sample disables Windows Update rather than modifying any other Windows Update settings. Therefore, we monitor the registry values listed in Table 6.19 for modifications. In this listing, we use the expression 'HKCU' to refer to 'HKU\[user id]'.

Registry subkey	**Registry value name**	**Percentage**
HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate\AU	NoAutoUpdate	74.09%
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer	NoWindowsUpdate	11.47%
HKCU\Software\Policies\Microsoft\Windows\Windows Update	NoWindowsUpdate	10.30%
HKCU\Software\Policies\Microsoft\Windows\Windows Update	NoAutoUpdate	10.30%
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\WindowsUpdate	DisableWindowsUpdateAccess	3.99%
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer	NoWindowsUpdate	3.41%
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\System	DisableWindowsUpdateAccess	2.98%
HKCU\Software\Policies\Microsoft\Windows\WindowsUpdate\AU	NoAutoUpdate	2.47%
HKCU\Software\Policies\Microsoft\Windows\Windows Update	NoWindowsUpdate	1.60%
HKCU\Software\Policies\Microsoft\Windows\Windows Update	NoAutoUpdate	1.60%
HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer	NoWindowsUpdate	0.22%
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\System	NoWindowsUpdate	0.07%

Table 6.19: Registry values, containing Windows Update settings, that are monitored for modification.

In addition to that, Windows Update can be deactivated by setting another IP for the domain http://windowsupdate.microsoft.com in the file 'C:\Windows\system32\drivers\etc\

hosts'. Currently, we cannot check for this behavior, because we do not parse the hosts file after analysis to include the modifications into the analysis report. But this enhancement is planned for a future release of Anubis.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that disable Windows Update.
- **Samples that disable Windows Update**: This graph gives the number of different samples that disable Windows Update. This graph is not visible by default.

In total, we identified 1 400 samples that disable the Windows Update. This represents 0.08% of all samples analyzed. We would have expected a higher share of samples that disable the Windows Update as many malware samples rely on vulnerabilities of Windows or the Internet Explorer for propagation. With Windows Update enabled, these vulnerabilities might be fixed with the next update. But more importantly, disabling Windows Update prevents the removal of malware by new versions of Windows Malicious Software Removal Tool.

In Table 6.19, we present the share samples for each monitored registered value. The most popular location to disable Windows Update clearly is `[HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate\AU] NoAutoUpdate`. Additionally, it is noticeable that 18.37% of the samples that disable Windows Update modify at least two values to accomplish the task.

6.33 Installation of Browser Helper Objects for the Internet Explorer

The chart presented in Figure 6.39 shows the samples that install Browser Helper Objects (BHO) for the Internet Explorer in 2009. This chart is available in monthly and daily intervals. The sample's analysis end determines its grouping.

A BHO is a COM object that is loaded automatically by the Internet Explorer or the Windows Explorer. It is used to customize the browser and extend the browser's functionality [34]. To identify a sample that installs a BHO, we simply check whether it creates a subkey in the registry key '`HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\`'. This subkey contains the CLSID that points to the COM class object, which is loaded by the Internet Explorer at last.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that install a Browser Helper Object.
- **Samples that install Browser Helper Objects**: This graph gives the number of samples that install a Browser Helper Object. This graph is hidden by default.

Altogether, we have observed 26 000 samples that install BHOs. This number represents a percentage of 1.42%. Most of these samples install only one BHO.

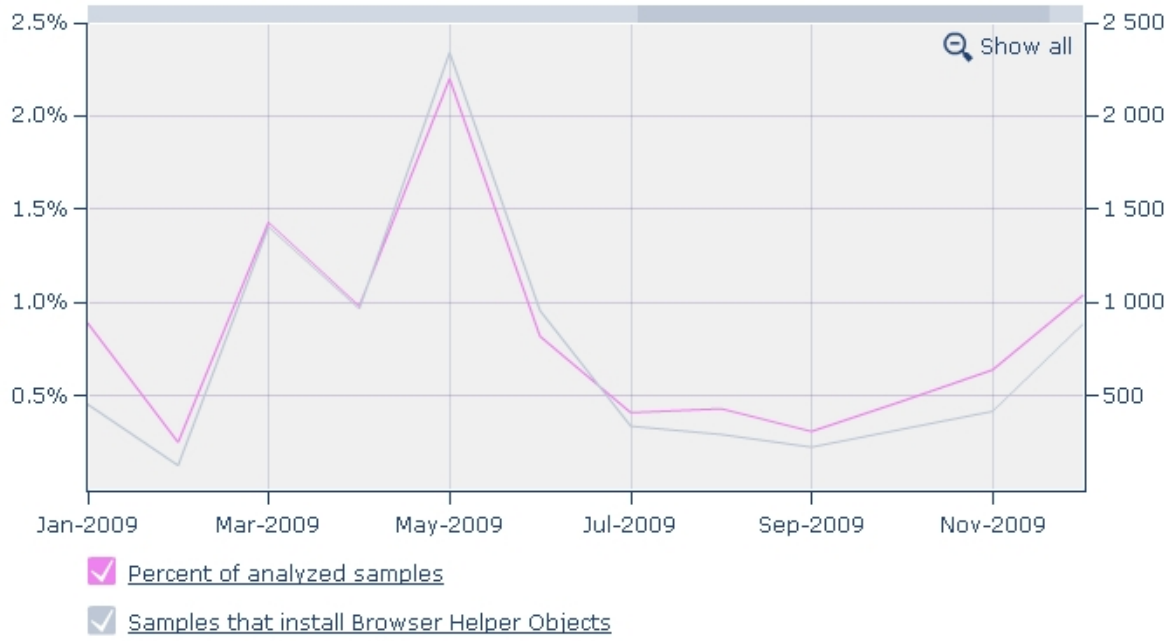


Figure 6.39: Monthly installations of Browser Helper Objects for the Internet Explorer

38.73% of the BHOs add a name to the newly installed BHO. Table 6.20 lists the 5 most popular names for BHOs. Most of these BHOs actually get a name that sounds innocent. Interestingly though, some BHO authors are perky and describe the BHOs assignment with names like 'AdPopup'.

BHO name	**Percentage**
XML module	9.41%
AdPopup	3.25%
LPVideoPlugin	2.02%
PK IE Plugin	1.18%
CodecPlugin Class	0.98%

Table 6.20: Most common names of BHOs.

6.34 Installation of Toolbars for the Internet Explorer

The chart in Figure 6.40 represents the samples that install toolbars for the Internet Explorer in 2009. It can be viewed in either monthly or daily intervals. The analysis end determines the grouping of the samples.

Toolbars are child windows that are displayed in the Internet Explorer or the Windows Explorer and are loaded automatically if either is started. It can display information and interact

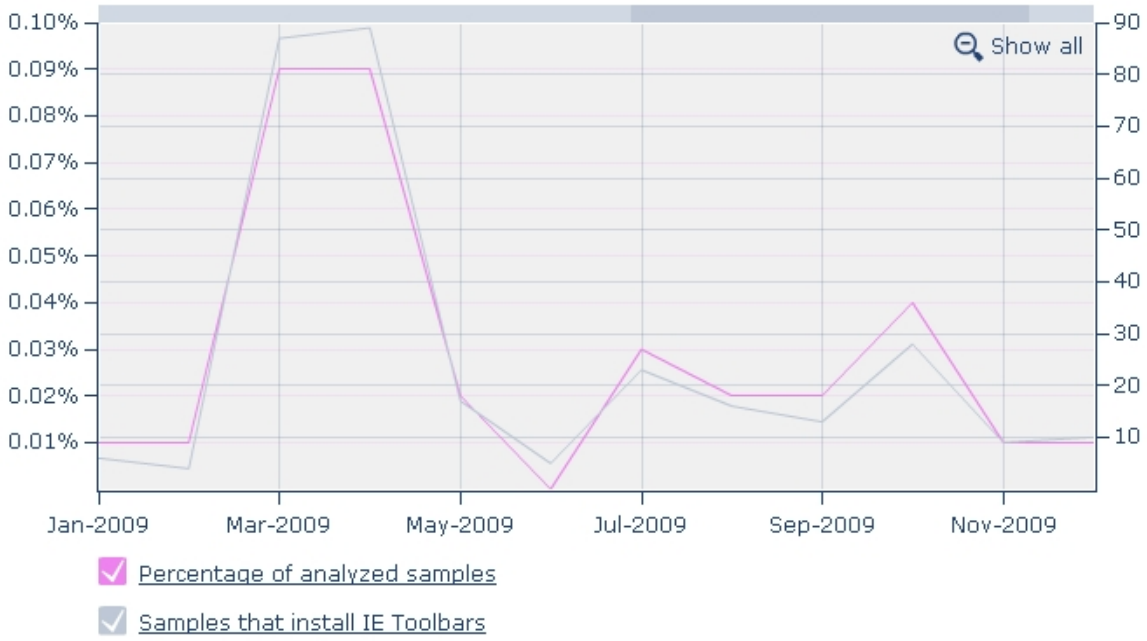


Figure 6.40: Monthly installations of toolbars for the Internet Explorer

with the user [10]. The CLSIDs of toolbars are stored as names with an empty value in the registry key '`HKLM\Software\Microsoft\Internet Explorer\Toolbar\`'. We monitor this key for modifications of the values to identify samples that install toolbars.

- **Percentage of analyzed samples**: This graph gives the percentage of analyzed samples that install toolbars for the Internet Explorer.
- **Samples that install IE Toolbars**: This graph represents the number of samples that installs toolbars for the Internet Explorer. It is not visible by default.

Altogether, Anubis detected 700 installations of toolbars for the Internet Explorer. This represents 0.04% of the analyzed samples. Usually, a sample installs only a single toolbar.

In Table 6.21, we list the most common CLSIDs that are registered as a toolbar for the Internet Explorer. Additionally, we extracted the according name of its COM object.

6.35 GUI Windows

The chart in Figure 6.41 represents the samples that create graphical user interface (GUI) windows in 2009. This chart is available in monthly or daily intervals. The sample's analysis end determines its grouping.

- **Percentage of analyzed samples**: This graph shows the percentage of analyzed samples that create GUI windows.

CLSID	DLL name	Percentage
B580CF65-E151-49C3-B73F-70B13FCA8E86	BaiduBar.dll	9.97%
74DD705D-6834-439C-A735-A6DBE2677452	VSAdd-in.dll	3.69%
C1B4DEC2-2623-438e-9CA2-C9043AB28508	Bar888.dll	3.55%
07B18EA9-A523-4961-B6BB-170DE4475CCA	MWSBAR.DLL	3.01%
0A1230F1-EB52-4CA3-9D34-DE2ABC2EED35	ToolBand.dll	2.19%

Table 6.21: Most common CLSIDs and DLLs for toolbars.

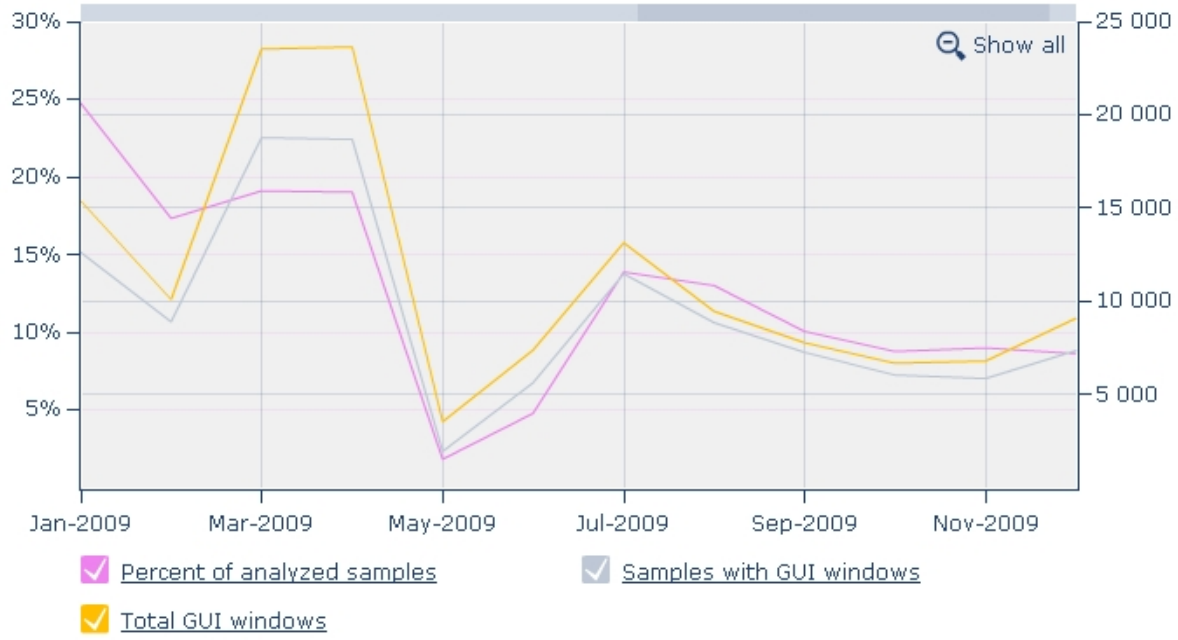


Figure 6.41: Monthly graphical user interface windows

- **Samples with GUI windows**: This graph represents the number of samples that create GUI windows. It is hidden by default.

- **Total GUI windows**: This graph gives the number of GUI windows that have been created by all samples in the specified interval. This graph is not visible by default.

Altogether, 330 000 samples created at least one GUI windows. This number represents 18.10% of all samples analyzed. In total, we recorded 680 000 GUI windows.

27.69% of these samples create a window containing an ‘OK’ button and 4.03% produce a window with ‘yes’ and ‘no’ buttons. The remaining windows are mostly more complex.

8.17% of these samples create GUI windows that refer to some installation process. Their title contains the words ‘setup’ or ‘installation’. Another 9.78% produce an error or warning message.

Interestingly, 8.81% of the samples that create GUI windows claim that they cannot locate a component. 5.25% report that they cannot find `cygwin1.dll` and thus cannot start the application. This DLL belongs to Cygwin and emulates the Linux API [12]. The remaining components are very diverse. A manual spot check reveals that these windows have been created on behalf of `csrss.exe`. Most of these samples do not perform any action at all, which confirms the suspicion that this error messages is not a fake.

During the analysis GUI windows are automatically closed after a short time period. This leads to numerous confirmation popups with the text 'Are you sure you want to quit xyz Setup?' with 'xyz' representing any application name. Most of these application names are related to porn like 'sexvid', 'pornvid' or 'deeporn'. Even though this pattern applies to 10.43% of the GUI windows that we captured, only 1.94% of the samples that actually produce GUI windows actually use such an installer that requires user interaction.

6.36 Output Streams

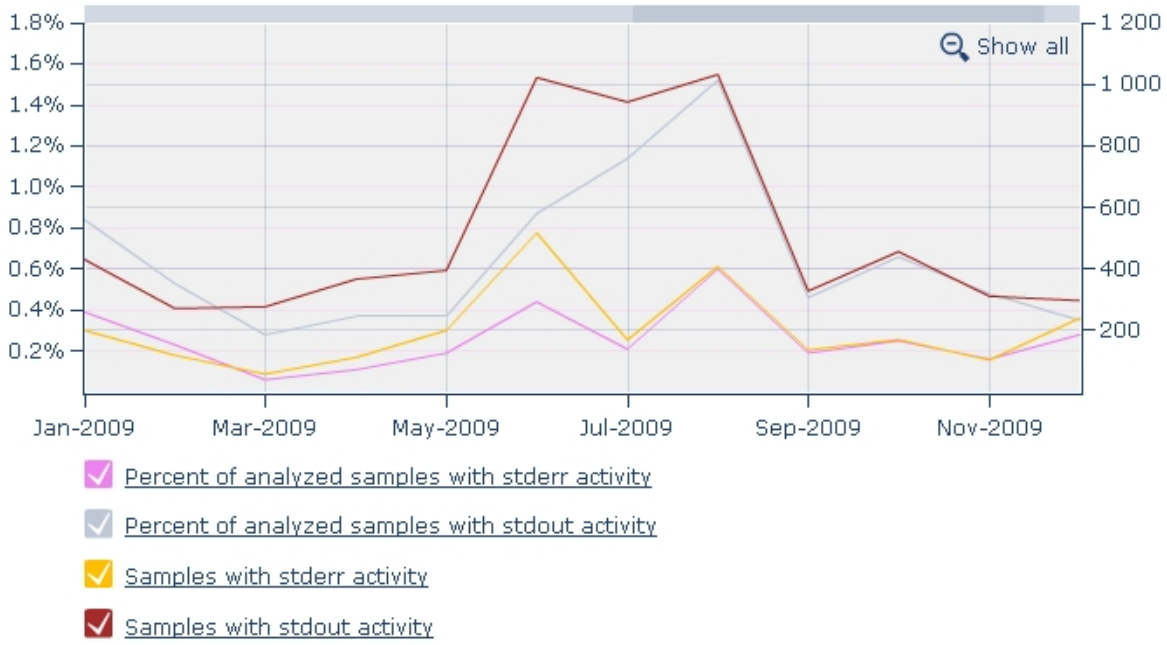


Figure 6.42: Monthly output streams

The chart illustrated in Figure 6.42 shows the samples that report messages to the standard out stream (`stdout`) or the standard error stream (`stderr`) in 2009. It is available in daily and monthly intervals. The analysis end of a sample determines its grouping.

- **Percentage of analyzed samples with stderr activity**: This graph represents the percentage of analyzed samples that write messages to `stderr`.

- **Percentage of analyzed samples with stdout activity**: This graph gives the percentage of analyzed samples that write messages to `stdout`.

- **Samples with stderr activity**: This graph shows the number of samples that write error messages to `stderr`. This graph is not visible by default.

- **Samples with stdout activity**: This graph represents the number of samples that report to `stdout`. It is hidden by default.

In total, 6 600 samples report error messages to `stderr`. This number represents 0.36% of all samples analyzed. 0.70% of the samples, 13 000 in numbers, report messages to `stdout`. 0.09% of the samples send messages to both output streams.

25.45% of the samples that write to `stdout` produce an output that contains an usage message. Another 20.08% include some kind of copyright message in the output and 6.03% of the messages to `stdout` even contain an email address, presumably the address of the component's author.

42.36% of the samples that write messages to `stderr` report some Java related error, most of them miss `java.dll` or require a version of Java prior to 1.6, which is installed on Anubis.

6.37 Service Installation

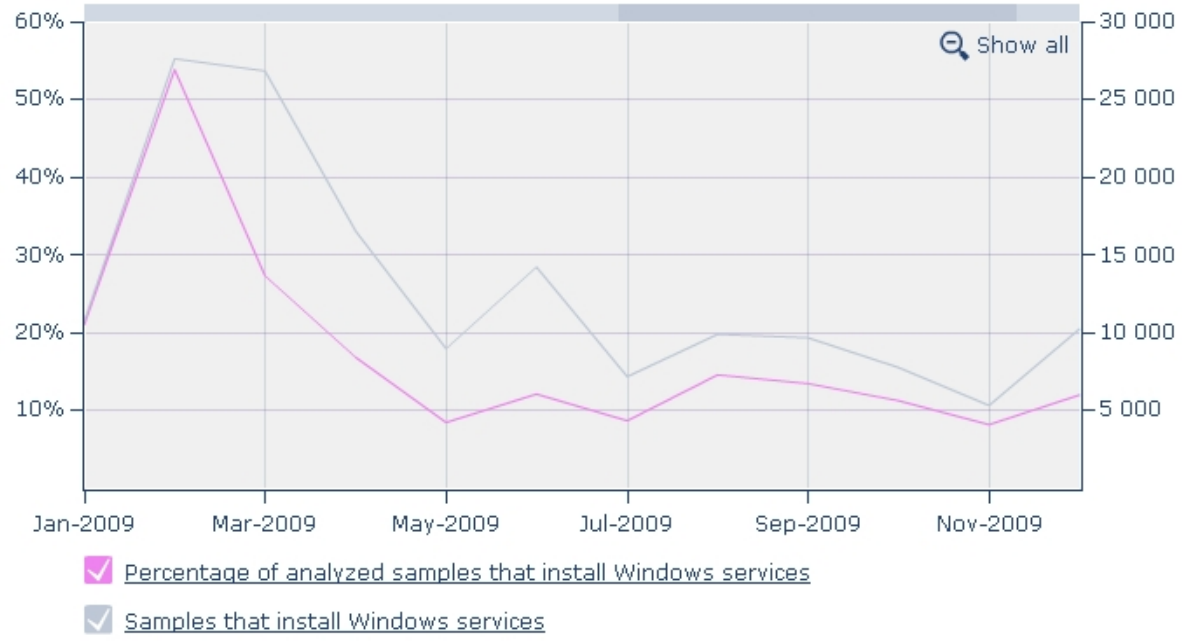


Figure 6.43: Monthly service installation

The chart illustrated in Figure 6.43 shows the samples that install a Windows service. This chart can be viewed with either monthly or daily intervals. A sample's analysis end determines its grouping.

Although Anubis monitors the `CreateService` function that adds a service to the service control manager's database, a service can be created manually by creating the necessary registry values in a subkey of '`HKLM\System\CurrentControlSet\Services\`'. For this reason, we consider a sample creating a service as well, if it creates a subkey in this registry key.

- **Percentage of analyzed samples that install Windows services**: This graph represents the percentage of analyzed samples that install services.
- **Samples that install Windows services**: This graph gives the number of samples that install a Windows service. This graph is hidden by default.

In total, Anubis detected 284 000 samples that install a service. This number represents 15.13% of the samples.

Service Name	Percentage
MSWindows	72.17%
Kisstusb	1.90%
NapAgent	1.89%
GrayPigeon_Hacker.com.cn	0.75%
WinSock2	0.40%

Table 6.22: Most stopped Windows services.

In Table 6.22, we list the most popular names of services that have been installed. By far the most prominent name is '`MSWindows`'. Once more, the polymorphic capabilities of Allaple have a huge impact on a statistic as it uses this name for the installation of a service. The remaining 27.83% of the services have more than 14 500 different names. Skimming some of these names, we conclude that a fair share of these samples tries to give names to their samples that sound innocently, like '`Windows Hosts Controller`', '`EventLog`' or '`win updates`'.

6.38 Service Stop

The chart in Figure 6.44 describes the samples that stop a Windows service in 2009. This chart is available in monthly and daily intervals. The samples analysis end determines its grouping.

Anubis monitors the control codes that are sent to the service control manager. The control code '`SERVICE_CONTROL_STOP`' causes the specified service to stop. In this statistic, we include only samples that send this control code to the service control manager.

- **Percentage of analyzed samples**: This graph gives the percentage of analyzed samples that stop a Windows service.

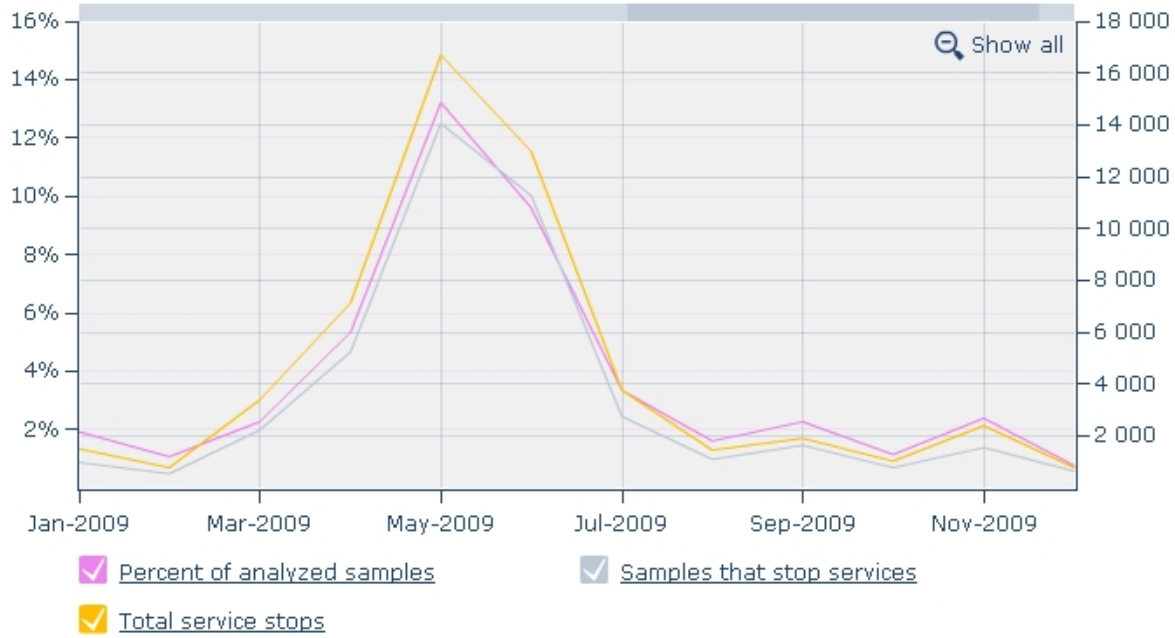


Figure 6.44: Monthly service stop

- **Samples that stop services**: This graph represents the number of samples that stop a Windows service. This graph is hidden by default.

- **Total servce stops**: This graph shows the total number of services that have been stopped. This chart is not visible by default.

In total, Anubis analyzed 64 000 samples that stop 84 000 services. This number represents 3.43% of all samples analyzed that stop a service.

In Table 6.23, we list the services that are stopped most often. The majority of samples that stop a service stop the Windows Security Center. By stopping this service, the user will not be notified anymore, if the firewall or the antivirus software is disabled. Changes to the Windows Automatic Update settings would be reported as well. Interestingly, seven of the top ten of stopped services are native to Windows XP [23]. They all provide functionality that directly improves the security of the system or can be used by third party software to improve the system's security.

6.39 Driver Load

The chart in Figure 6.45 gives an overview of the samples that load a kernel driver in 2009. This chart can be viewed in monthly and daily intervals. A samples analysis end determines its group affiliation.

Service Name	Display Name	Percentage
WSCSVC	Security Center	39.67%
spooler	Print Spooler	17.02%
SharedAccess	Windows Firewall/Internet Connection Sharing	15.00%
beep	Beep	7.30%
KPDrvLN	KPDrvLN	3.29%
ALG	Application Layer Gateway Service	2.07%
RemoteRegistry	Remote Registry	2.01%
wuauserv	Automatic Updates	2.01%
NetApi00	NetApi00	1.13%
Schedule	Task Scheduler	1.00%

Table 6.23: Most stopped Windows services.

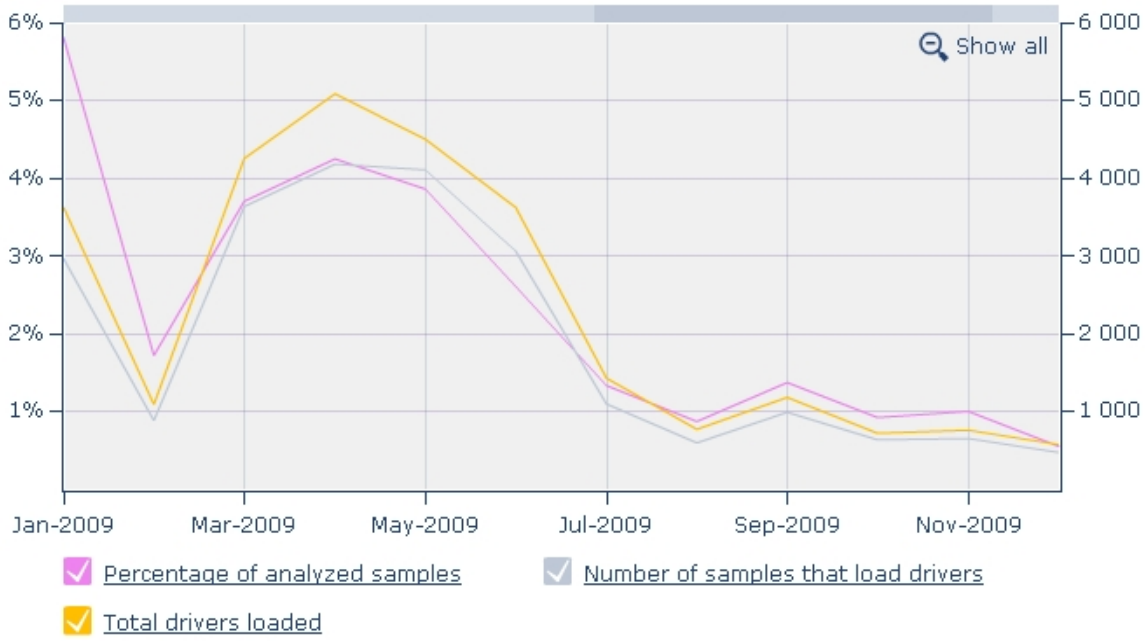


Figure 6.45: Monthly driver load

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that load a driver.
- **Number of samples that load drivers**: This graph shows the number of samples that load a driver. This graph is hidden by default.
- **Total drivers loaded**: This graph gives the total number of drivers that have been loaded. It is not visible by default.

Altogether, 2.45% of the samples analyzed load a driver. This percentage represents 46 000 samples that loaded 58 000 drivers during analysis.

Driver Name	Percentage
UACd.sys	11.40%
Kisstusb	9.26%
Beep	8.47%
KPDrvLN	4.82%
TDSSserv.sys	4.70%
AsyncMac	2.64%
IpFilterDriver	2.52%
WS2IFSL	1.81%
Driver	1.77%
NetApi00	1.68%

Table 6.24: Most loaded drivers.

Table 6.24 shows the names of the drivers that are most often loaded. Interestingly, almost 15% of the drivers loaded are native to Windows like 'Beep', 'AsyncMac', 'aec' or 'Secdrv'. Further investigation reveals that the driver itself or the path that points to a driver has been modified by the malware before loading the Windows driver.

6.40 Process Creation

The chart presented in Figure 6.46 describes the samples that create processes in 2009. This chart is available with daily and monthly intervals. The assorting utilizes a samples analysis end.

- **Percentage of analyzed samples**: This graph shows the percentage of analyzed samples that create processes.
- **Samples that create processes**: This graph represents the number of samples that create processes. This graph is hidden by default.
- **Total processes created**: This graph gives the total number of process created. It is not visible by default.

In total, 796 000 samples create processes. This number represents 41.92% of all samples analyzed. On average, each of these processes creates 5.6 processes in the four minutes of its analysis.

Creating processes is very popular amongst malware. Almost half of our samples perform this task. Many rely on functionality provided by Windows executables and others download or unpack additional executables that they execute. Table 6.25 lists the most prominent processes that have been executed. Apart from `urdvxc.exe` that is called by Allaple, all the remaining processes in this listing are native to Windows.

Figure 6.46: Monthly process creation

Process	Percentage
C:\Windows\System32\urdvxc.exe	21.62%
C:\Windows\System32\cmd.exe	17.59%
C:\Program Files\Internet Explorer\iexplore.exe	14.31%
C:\Windows\System32\drwtsn32.exe	13.76%
C:\Windows\System32\dwwin.exe	9.66%
C:\Windows\System32\net.exe	5.82%
C:\Windows\System32\net1.exe	5.52%
C:\Windows\System32\svchost.exe	4.94%
C:\Windows\System32\regsvr32.exe	3.83%
C:\Windows\System32\sc.exe	3.39%

Table 6.25: Most often created processes.

It is noticeable that 13.76% of the samples that create processes invoke `drwrsn32.exe`. This process is a error debugger and collects information about program crashes [1]. Creation of this process usually indicates that something went wrong during the execution of the sample and that it crashed.

6.41 Process Termination

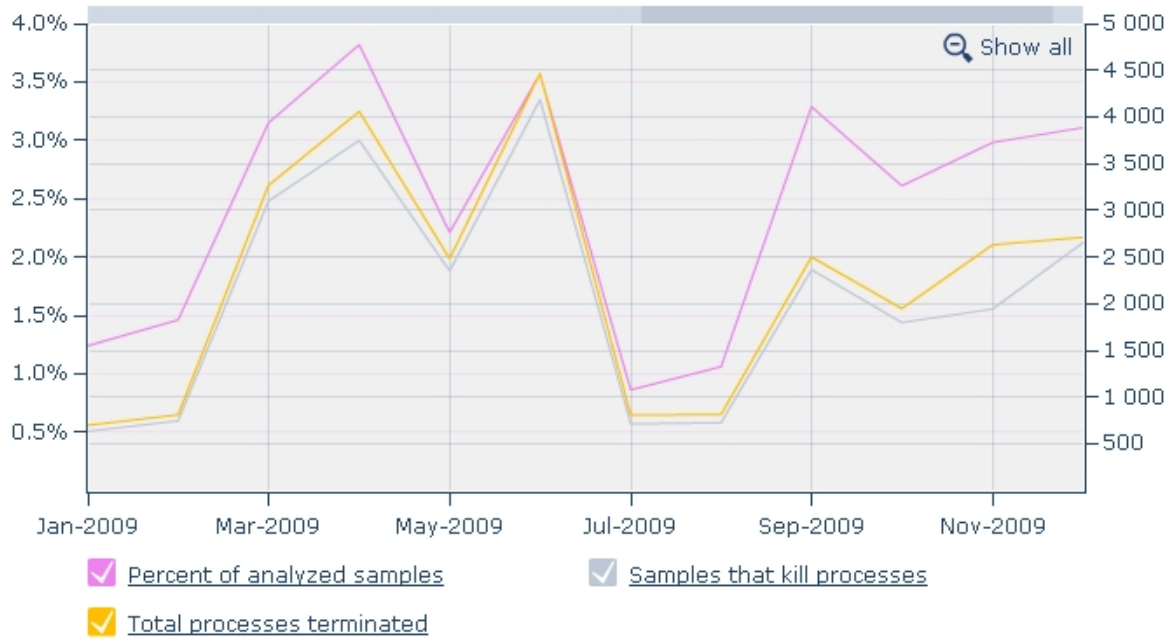


Figure 6.47: Monthly process termination

The chart in Figure 6.47 shows the samples that terminate processes in 2009. The analysis end of a sample is used to group it. This chart can be viewed in monthly and daily intervals.

- **Percentage of analyzed samples**: This graph shows the percentage of analyzed samples that kill a process.
- **Samples that kill processes**: This graph represents the number of samples that terminate a process. This graph is hidden by default.
- **Total processes terminated**: This graph gives the total number of processes terminated. This graph is not visible by default.

In total, we observed 45 000 samples that terminate processes. This value represents 2.39% of the samples. Altogether, 51 000 processes were killed.

Killing processes is by far not as popular as creating processes. This may be due to the fact that we do not have running any antivirus products in the analysis environment. Most probably we would encounter more process terminations if some of the running processes were system security related processes. But a process that does not run cannot be terminated, thus we cannot observe this behavior. 1.45% of the samples that kill a process terminate the Windows Security Center, which notifies the user if any changes to antivirus settings, firewall settings or Windows update settings are performed.

6.42 Write to Foreign Memory Area

Figure 6.48: Monthly writes to foreign memory area

The chart illustrated in Figure 6.48 gives an overview of the samples that write to foreign memory area in 2009. This chart is available in daily and monthly intervals. A sample's analysis end determines its grouping.

- **Percentage of analyzed samples**: This graph represents the percentage of analyzed samples that write to foreign memory areas.
- **Samples that write to foreign memory area**: This graph gives the number of samples that write to foreign memory areas. This graph is hidden by default.
- **Total foreign memory writes**: This graph shows the total number of processes that a sample requests to write to foreign memory areas. This graph is not visible by default.

In total, Anubis observed 732 000 samples that write to foreign memory areas. This number represents 38.54% of all samples analyzed.

In Table 6.26, we list the most popular processes that write to foreign memory areas. Note that the percentage exceeds 100% as it is common for a sample to have multiple executables to write to foreign memory areas. Actually, each of these samples instructs 3.23 processes on average to write to foreign memory areas. This listing reveals that it is very popular amongst malware to request native Windows processes to write to foreign memory areas. In this statistic again, we can see the marks of Allaple's process `urdvxc.exe`.

Process	Percentage
C:\Windows\System32\cmd.exe	23.64%
C:\Windows\System32\urdvxc.exe	18.39%
C:\Program Files\Internet Explorer\iexplore.exe	15.28%
C:\Windows\explorer.exe	13.61%
C:\Windows\System32\svchost.exe	8.76%
C:\Windows\System32\dwwin.exe	7.90%
C:\Windows\System32\spoolsv.exe	7.64%
C:\Windows\System32\winlogon.exe	6.78%
C:\Windows\System32\wscntfy.exe	6.69%
C:\Windows\System32\ctfmon.exe	6.65%

Table 6.26: Most popular processes that write to foreign memory areas.

CHAPTER 7

Future Work

In this chapter, we highlight some properties of the statistics generation environment and Anubis in general that would lead to more thorough views of malware behavior or would increase the efficiency and effectiveness of the generation process and the storage of the analysis data.

7.1 Prospective Views of Malware Behavior

Of course, the list of statistics that represent high-level behavior of malware can be extended, but only to some extent. Some behavior cannot be visualized due to limitations of the available data. For example, we can see whether the file `C:\Windows\system32\drivers\etc\hosts` is being modified, but we do not know which domain and IP mappings have been set. In order to gain deeper insight into malware behavior, we could extend the analysis report and include this information. Since the `hosts` file is empty initially, we would simply have to parse it after analysis is complete to extract the modifications. This would allow us to conclude about the type of domains that are redirected and whether they are redirected to another server, possibly under control of the malware author, or whether these domains are blocked by redirecting them to `127.0.0.1`. Currently, this feature is being implemented and tested, so it will not take long until we can see first results.

INI files were used in the early days of Windows to store system configuration. In Windows XP these files are still in use and they are mapped to the registry but the modifications to these files are not propagated to the registry immediately. Even though we can see whether an INI file has been modified, the report does not provide additional information about the specific sections and parameters that have been modified in these files. However, by modifying INI files like `C:\Windows\system.ini`, system settings can be altered and malware could register components in order to be executed upon system reboot. Detailed information on the sections and parameters that have been modified in INI files that are part of the Windows installation will provide additional insight into malware behavior.

7.2 Modification to the Environment

Furthermore, it would be interesting to observe the interaction of the malware with third party security applications like antivirus programs or software firewalls. However, the environment would have to be configured in such a way that these applications do not interfere with the sample under analysis. It would be counterproductive if an antivirus program would hinder the sample from executing or if the software firewall would prevent network communication, as we could not monitor the intended behavior of malware. On the other hand, it would be informative to see whether sophisticated malware not only bypasses security restrictions set by the operating system but also additional restrictions set by third party applications.

In order to observe further interaction with the environment, we could fill the Windows address book with some bogus email contacts and place saved emails in the inbox and sent folder of Microsoft Outlook Express or another email client. Additionally, we could install popular instant messaging programs like Skype, ICQ or MSN and detect if malware tries to propagate itself over these channels or abuses these programs for spam. Finally, we could save user credentials to popular online portals like Facebook and MySpace or passwords for online banking in the Internet Explorer and observe whether malware steals these credentials or tries to use them instantly. One drawback of these environment modifications would be that they could be used by malware to detect Anubis and the malware under analysis could change its behavior as a consequence.

7.3 Opportunities for Improvement of Efficiency

Currently, the queries to calculate the statistics take less than 30 minutes daily. The main cause for long query runtime is the fact that we do not save shortcuts for registry keys in the analysis report. The registry hive `HKCU` is such a shortcut for `HKU\S-1-5-xx-xxxxxxxxxx-xxxxxxxxxx-xxxxxxxxx-xxxx\`. Since the user id in `HKU` is subject to randomization in order to prevent the detection of Anubis, querying for such a registry key is very expensive because the query has to be constructed like that:

```
...  WHERE key_name LIKE "HKU\\\\%\\\\..."
```

By replacing these registry keys with the short cut `HKCU`, the query would be simpler and run more efficiently, as it would not require the `LIKE` comparator anymore and could make full use of the index on the field that stores the registry key. Additionally, storage space would be saved because any registry key that is a subkey of `HKU` would only have to be stored once instead of once for each randomization of the Windows user ID.

7.4 Future Work on Chart Display

Recently, HTML5 gains more and more acceptance, as the most popular browsers start supporting this standard. Currently, amCharts, a Flash application, is responsible for the display of graphs and the interaction with the same. In order not to rely on the Flash plug-in, it would be beneficial

to replace the Flash application with a charting application that is based on HTML5 at some point in the future.

Although not primarily required by a charting application, another handy feature would be the opportunity to export charts as a vector graphic, for example in the SVG format. Even though this feature is not important for the web-based display of the charts, it would tremendously increase the quality of printed versions of the charts in publications.

APPENDIX A

Database Schema

address_scan (address_scan_id, *simple_db_report.result_id* ‘result_id’, a_count, remote_port, subnet, protocol)

device_control_communication (device_control_communication_id, *simple_db_report.result_id* ‘result_id’, *file_name.file_name_id* ‘file_name_id’, control_code, count)

directory_monitored (directory_monitored_id, *simple_db_report.result_id* ‘result_id’, *file_name.file_name_id* ‘directory’, notify_filter, watch_subtree)

dns_query (dns_query_id, *simple_db_report.result_id* ‘result_id’, dest_ip, dest_port, query_type, name, result, protocol, successful)

driver_loaded (driver_loaded_id, *simple_db_report.result_id* ‘result_id’, *file_name.file_name_id* ‘driver’)

driver_unloaded (driver_unloaded_id, *simple_db_report.result_id* ‘result_id’, *file_name.file_name_id* ‘driver’)

exception_occurred (exception_occurred_id, *simple_db_report.result_id* ‘result_id’, description, count)

file_created (file_created_id, *simple_db_report.result_id* ‘result_id’, *file_name.file_name_id* ‘file_name_id’, is_directory)

file_deleted (file_deleted_id, *simple_db_report.result_id* ‘result_id’, *file_name.file_name_id* ‘file_name_id’, is_directory)

file_modified (file_deleted_id, *simple_db_report.result_id* 'result_id', *file_name.file_name_id* 'file_name_id')

file_name (file_name_id, name)

file_read (file_read_id, *simple_db_report.result_id* 'result_id', *file_name.file_name_id* 'file_name_id')

file_renamed (file_renamed_id, *simple_db_report.result_id* 'result_id', *file_name.file_name_id* 'new_name', *file_name.file_name_id* 'old_name')

foreign_mem_area_read (foreign_mem_area_read_id, *simple_db_report.result_id* 'result_id', *process_executable_name.process_executable_name_id* 'process')

foreign_mem_area_write (foreign_mem_area_write_id, *simple_db_report.result_id* 'result_id', *process_executable_name.process_executable_name_id* 'process')

fs_control_communication (fs_control_communication_id, *simple_db_report.result_id* 'result_id', control_code, count, *file_name.file_name_id* 'file_name_id')

ftp_conversation (ftp_conversation_id, *simple_db_report.result_id* 'result_id', dest_ip, dest_port, user_name, user_name_ok, password, password_ok)

http_get_data (http_get_id, *simple_db_report.result_id* 'result_id', dest_ip, dest_port, host, req_num, client_request, client_request_version, client_headers, client_browser, server_response, server_headers, server_name, server_date, server_content_type, server_content_subtype, server_content_length, server_body)

http_head_data (http_head_id, *simple_db_report.result_id* 'result_id', dest_ip, dest_port, host, req_num, client_request, client_request_version, client_headers, client_browser, server_response, server_headers, server_name, server_date, server_content_type, server_content_subtype, server_content_length, server_body)

http_post_data (http_post_id, *simple_db_report.result_id* 'result_id', dest_ip, dest_port, host, req_num, client_request, client_request_version, client_content_type, client_content_subtype, client_content_length, client_headers, client_browser, client_browser, client_body, server_response, server_headers, server_name, server_date, server_content_type, server_content_subtype, server_content_length, server_body)

icmp_traffic (icmp_traffic_id, *simple_db_report.result_id* 'result_id', sent, got_reply, subnet)

ikarus (ikarus_id, ikarus)

irc_channel (irc_channel_id, *irc_conversation.irc_conversation_id* 'irc_conversation_id', channel_name, channel_password, channel_topic)

irc_conversation (irc_conversation_id, *simple_db_report.result_id* 'result_id', dest_ip, dest_port, server_password, nick_name, user_name)

key_was_checked (key_was_checked_id, *simple_db_report.result_id* 'result_id', key, count)

link_created (link_created_id, *simple_db_report.result_id* 'result_id', *file_name.file_name_id* 'existing_file', *file_name.file_name_id* 'link_name')

mutex_created (mutex_created_id, *simple_db_report.result_id* 'result_id', *mutex_name.mutex_name_id* 'mutex_name_id')

mutex_name (mutex_name_id, name)

opened_port (opened_port_id, *simple_db_report.result_id* 'result_id', port_type, port_no)

popup_window (popup_window_id, *simple_db_report.result_id* 'result_id', window_name, window_text)

port_scan (port_scan_id, *simple_db_report.result_id* 'result_id', p_count, remote_ip, protocol)

privmsg_to_channel (privmsg_to_channel_id, *irc_conversation.irc_conversation_id* 'irc_conversation_id', channel, priv_msg)

privmsg_to_user (privmsg_to_user_id, *irc_conversation.irc_conversation_id* 'irc_conversation_id', user, priv_msg)

process_created (process_created_id, *simple_db_report.result_id* 'result_id', *file_name.file_name_id* 'process_dir', *process_executable_name.process_executable_name_id* 'process_executable_name_id', *process_parameters.process_parameters_id* 'process_parameters_id')

process_executable_name (process_executable_name_id, executable_name)

process_killed (process_killed_id, *simple_db_report.result_id* 'result_id', *process_executable_name.process_executable_name_id* 'process')

process_parameters (process_parameters_name_id, parameters)

registry_entry (registry_entry_id, *registry_key_name.registry_key_name_id*

'registry_key_name_id', *registry_value_name.registry_value_name_id* 'registry_value_name_id')

registry_key_created (registry_key_created_id, *simple_db_report.result_id* 'result_id', *registry_key_name.registry_key_name_id* 'registry_key_name_id')

registry_key_created_or_opened (registry_key_created_or_opened_id, *simple_db_report.result_id* 'result_id', *registry_key_name.registry_key_name_id* 'registry_key_name_id')

registry_key_deleted (registry_key_deleted_id, *simple_db_report.result_id* 'result_id', *registry_key_name.registry_key_name_id* 'registry_key_name_id')

registry_key_monitored (registry_key_monitored_id, *simple_db_report.result_id* 'result_id', *registry_key_name.registry_key_name_id* 'registry_key_name_id', watch_subtree, notify_filter, count)

registry_key_name (registry_key_name_id, key_name)

registry_value (registry_value_id, value)

registry_value_deleted (registry_value_deleted_id, *simple_db_report.result_id* 'result_id', *registry_entry.registry_entry_id* 'registry_entry_id')

registry_value_modified (registry_value_modified_id, *simple_db_report.result_id* 'result_id', *registry_entry.registry_entry_id* 'registry_entry_id', *registry_value.registry_value_id* 'registry_value_id')

registry_value_name (registry_value_name_id, value_name)

registry_value_read (registry_value_read_id, *simple_db_report.result_id* 'result_id', *registry_entry.registry_entry_id* 'registry_entry_id', *registry_value.registry_value_id* 'registry_value_id', times)

section_object_created (section_object_created_id, *simple_db_report.result_id* 'result_id', *file_name.file_name_id* 'file_name_id')

service_changed (service_changed_id, *simple_db_report.result_id* 'result_id', *service_name.service_name_id* 'service_name_id')

service_control_code (service_control_code_id, *simple_db_report.result_id* 'result_id', *service_name.service_name_id* 'service_name_id', control_code)

service_created (service_created_id, *simple_db_report.result_id* ‘result_id’, *service_name.service_name_id* ‘service_name_id’, *file_name.file_name_id* ‘service_path_id’, type)

service_deleted (service_deleted_id, *simple_db_report.result_id* ‘result_id’, *service_name.service_name_id* ‘service_name_id’)

service_name (service_name_id, name)

service_started (service_started_id, *simple_db_report.result_id* ‘result_id’, *service_name.service_name_id* ‘service_name_id’)

sigbuster (sigbuster_id, sigbuster)

simple_db_report (*result.result_id* ‘result_id’, profile, listens_on_port, *sigbuster.sigbuster_id* ‘sigbuster_id’, *ikarus.ikarus_id* ‘ikarus_id’, stdout, stderr, termination_reason)

smtp_attached_file (smtp_attached_file_id, *smtp_conversation.smtp_conversation_id* ‘smtp_conversation_id’, filename, content_type, content_subtype)

smtp_conversation (smtp_conversation_id, *simple_db_report.result_id* ‘result_id’, subject, content, sender, dest_ip)

smtp_recipient (smtp_recipient_id, *smtp_conversation.smtp_conversation_id* ‘smtp_conversation_id’, recipient_name, email_name, email_domain)

tcp_connection_attempt (tcp_connection_attempt_id, *simple_db_report.result_id* ‘result_id’, dest_ip, dest_port, state)

unknown_tcp_traffic (unknown_tcp_traffic_id, *simple_db_report.result_id* ‘result_id’, dest_ip, dest_port, org_bytes_sent, res_bytes_sent, state)

unknown_udp_traffic (unknown_udp_traffic_id, *simple_db_report.result_id* ‘result_id’, dest_ip, dest_port, org_bytes_sent, res_bytes_sent, state)

APPENDIX B

Autostart Registry Keys

Registry key
HKCU\Control Panel\Desktop
HKCU\Software\Classes\%\shell\%\command
HKCU\Software\Classes\%\shellex\
HKCU\Software\Classes\.%\ShellNew
HKCU\Software\Microsoft\Active Setup\Installed Components
HKCU\Software\Microsoft\Command Processor
HKCU\Software\Microsoft\CTF\LangbarAddin
HKCU\Software\Microsoft\Internet Explorer\Desktop\Components
HKCU\Software\Microsoft\Internet Explorer\Explorer Bars
HKCU\Software\Microsoft\Internet Explorer\Extensions
HKCU\Software\Microsoft\Internet Explorer\URLSearchHooks
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\Run
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\ RunOnceEx
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ FileExts\
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ MountPoints

Registry key
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ ShellIconOverlayIdentifiers
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
HKCU\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\System
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKCU\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved
HKCU\Software\Microsoft\Windows\CurrentVersion\ ShellServiceObjectDelayLoad
HKCU\Software\Policies\Microsoft\Windows\System\Scripts
HKLM\Software\Classes\%\shell\%\Command
HKLM\Software\Classes\%\shellex\
HKLM\Software\Classes\.%\ShellNew
HKLM\Software\Classes\Protocols\Filter
HKLM\Software\Classes\Protocols\Handler
HKLM\Software\Microsoft\Active Setup\Installed Components
HKLM\Software\Microsoft\Command Processor
HKLM\Software\Microsoft\CTF\LangbarAddin
HKLM\Software\Microsoft\Internet Explorer\Explorer Bars
HKLM\Software\Microsoft\Internet Explorer\Extensions
HKLM\Software\Microsoft\Internet Explorer\Toolbar
HKLM\Software\Microsoft\Shared Tools\MSConfig\startupfolder
HKLM\Software\Microsoft\Shared Tools\MSConfig\startupreg
HKLM\Software\Microsoft\Windows NT\CurerntVersion\Windows
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Accessibility\ Utility Manager
HKLM\Software\Microsoft\Windows NT\CurrentVersion\AeDebug
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Drivers32
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\Run
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\RunOnce

Registry key
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\ RunOnceEx
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ AutoplayHandlers\Handlers
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ MyComputer\BackupPath
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ MyComputer\ChkDskPath
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ MyComputer\cleanuppath
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ MyComputer\DefragPath
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ SharedTaskScheduler
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ ShellExecuteHooks
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ ShellIconOverlayIdentifiers
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System
HKLM\Software\Microsoft\Windows\CurrentVersion\Run
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnceEx
HKLM\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved
HKLM\Software\Microsoft\Windows\CurrentVersion\ ShellServiceObjectDelayLoad
HKLM\Software\Policies\Microsoft\Windows\System\Scripts
HKLM\System\CurrentControlSet\Control\BootVerificationProgram
HKLM\System\CurrentControlSet\Control\Lsa
HKLM\System\CurrentControlSet\Control\NetworkProvider\Order
HKLM\System\CurrentControlSet\Control\Print\Monitors

Registry key
HKLM\System\CurrentControlSet\Control\SafeBoot\Option
HKLM\System\CurrentControlSet\Control\SecurityProviders
HKLM\System\CurrentControlSet\Control\Session Manager
HKLM\System\CurrentControlSet\Control\Terminal Server\Wds\rdpwd
HKLM\System\CurrentControlSet\Control\WOW
HKLM\System\CurrentControlSet\Services\

Table B.1: Autostart registry keys.

Bibliography

[1] Description of the Dr. Watson for Windows (Drwtsn32.exe) Tool. http://support.microsoft.com/kb/308538, 2007.

[2] Microsoft Windows Does Not Disable AutoRun Properly. http://www.us-cert.gov/cas/techalerts/TA09-020A.html, 2009.

[3] A definition of the Run keys in the Windows XP registry. http://support.microsoft.com/kb/314866, 2010.

[4] amCharts. http://www.amcharts.com/, 2010.

[5] amCharts - Line & Area Settings Reference. http://www.amcharts.com/docs/v.1/bundle/settings/line_area, 2010.

[6] amCharts - Pie & Donut Settings Reference. http://www.amcharts.com/docs/v.1/bundle/settings/pie_donut, 2010.

[7] Anubis: Analyzing Unknown Binaries. http://anubis.iseclab.org/, 2010.

[8] Art of file - graphical interpretation of a file. http://gynvael.coldwind.pl/?id=199, 2010.

[9] Changing the Radmin Server port. http://www.radmin.com/support/kb/?ID=3270, 2010.

[10] Creating Custom Explorer Bars, Tool Bands, and Desk Bands. http://msdn.microsoft.com/en-us/library/cc144099%28VS.85%29.aspx, 2010.

[11] CWSandbox. http://www.cwsandbox.org/, 2010.

[12] Cygwin Information and Installation. http://www.cygwin.com/, 2010.

[13] Definition of the RunOnce Keys in the Registry. http://support.microsoft.com/kb/137367, 2010.

[14] FIRE: FInding RoguE Networks. http://maliciousnetworks.org/, 2010.

[15] Google Safe Browsing API. http://code.google.com/apis/safebrowsing/, 2010.

[16] How MySQL Uses Indexes. http://dev.mysql.com/doc/refman/5.5/en/mysql-indexes.html, 2010.

[17] How to Run Automatic Commands When Starting in MS-DOS Mode. http://support.microsoft.com/kb/141308, 2010.

[18] IANA | MIME Media Types. http://www.iana.org/assignments/media-types/, 2010.

[19] Nepenthes. http://nepenthes.carnivore.it/, 2010.

[20] Port Numbers. http://www.iana.org/assignments/port-numbers, 2010.

[21] VirusTotal - Free Online Virus and Malware Scan. http://www.virustotal.com/, 2010.

[22] Wepawet. http://wepawet.iseclab.org/, 2010.

[23] Windows XP x86 (32-bit) Service Pack 3 Service Configurations. http://www.blackviper.com/WinXP/servicecfg.htm, 2010.

[24] WOMBAT: Worldwide Observatory of Malicious Behaviors and Attack Threats. http://wombat-project.eu/, 2010.

[25] Kulsoom Abdullah, Chris Lee, Gregory Conti, and John A. Copeland. Visualizing network data for intrusion detection. In *Proceedings of the 2002 IEEE*, pages 100–108. IEEE, 2002.

[26] Richard Adhikari. Spammers Ramp Up Short-Lived Web Sites. http://www.internetnews.com/security/article.php/3798666, 2009.

[27] Wolfgang Aigner, Silvia Miksch, Wolfgang Muller, Heidrun Schumann, and Christian Tominski. Visual methods for analyzing time-oriented data. *IEEE Transactions on Visualization and Computer Graphics*, 14:47–60, 2007.

[28] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. Visualizing time-oriented data-a systematic view. *Comput. Graph.*, 31(3):401–409, 2007.

[29] Andrei Arion, Angela Bonifati, Ioana Manolescu, and Andrea Pugliese. XQueC: A query-conscious compressed XML database. *ACM Trans. Internet Technol.*, 7(2):10, 2007.

[30] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. Insights into current malware behavior. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[31] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. Information visualization. pages 1–34, 1999.

[32] William S. Cleveland. *Visualizing Data*. Hobart Press, 1993.

[33] Alex Dragulescu. Malwarez. http://www.sq.ro/malwarez.php, 2010.

[34] Dino Esposito. Browser Helper Objects: The Browser the Way You Want It. http://msdn.microsoft.com/en-us/library/bb250436%28VS.85%29.aspx, 1999.

[35] Stephen Few. *Show me the numbers: Designing Tables and Graphs to Enlighten*. Analytics Press, 2004.

[36] Elia Florio. W32.Rahack.W - Technical Details. http://www.symantec.com/security_response/writeup.jsp?docid=2007-011509-2103-99&tabid=2, 2007.

[37] Andrew U. Frank. Different types of "Times" in GIS. *Spatial and temporal reasoning in geographic information systems*, pages 40–62, 1998.

[38] Paul Gauthier, Josh Cohen, Martin Dunsmuir, and Charles Perkins. Web Proxy Auto-Discovery Protocol. http://tools.ietf.org/html/draft-ietf-wrec-wpad-01, 1999.

[39] Steve Gibson. Port Authority Database - Port 139. http://www.grc.com/port_139.htm, 2010.

[40] Steve Gibson. Port Authority Database - Port 445. http://www.grc.com/port_445.htm, 2010.

[41] Fanglu Guo, Peter Ferrie, and Tzi-Cker Chiueh. A study of the packer problem and its solutions. In *RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*, pages 98–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[42] Jerry Honeycutt. *Microsoft Windows XP Registry Guide*. Microsoft Press, Redmond, WA, USA, 2002.

[43] Christian Huitema. The Windows XP/SP2 Firewall. http://www.huitema.net/sp2-firewall.asp, 2004.

[44] Sergey Ilyin. Testing of antivirus software for the detection of polymorphic viruses. http://www.anti-malware-test.com/?q=node/47, 2008.

[45] W. H. Inmon. *Building the Data Warehouse,3rd Edition*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[46] Hartmut Liefke and Dan Suciu. XMill: an efficient compressor for XML data. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 153–164, New York, NY, USA, 2000. ACM.

[47] Yarden Livnat, Jim Agutter, Shaun Moon, Robert F. Erbacher, and Stefano Foresti. A visual paradigm for network intrusion detection. In *In Proceedings of the 2005 IEEE Workshop on Information Assurance And Security*, pages 92–99. IEEE, 2005.

[48] Florian Mansmann, Fabian Fischer, Daniel A. Keim, and Stephen C. North. Visual support for analyzing network traffic and intrusion detection events using treemap and graph representations. In *CHiMiT '09: Proceedings of the Symposium on Computer Human Interaction for the Management of Information Technology*, pages 19–28, New York, NY, USA, 2009. ACM.

[49] Raffael Marty. *Applied Security Visualization*. Addison-Wesley Professional, 2008.

[50] Paolo Milani, Guido Salvaneschi, Clemens Kolbitsch, Christopher Kruegel, Engin Kirda, and Stefano Zanero. Identifying Dormant Functionality in Malware Programs. 2010.

[51] Wolfgang Müller and Heidrun Schumann. Visualization methods for time-dependent data-an overview. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, 2003.

[52] Matthias Nicola and Jasmi John. XML parsing: a threat to database performance. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 175–178, New York, NY, USA, 2003. ACM.

[53] Steven Noel, Michael Jacobs, Pramod Kalapa, and Sushil Jajodia. Multiple coordinated views for network attack graphs. In *VIZSEC '05: Proceedings of the IEEE Workshops on Visualization for Computer Security*, page 12, Washington, DC, USA, 2005. IEEE Computer Society.

[54] Andy Oppel. *Databases A Beginner's Guide*. McGraw-Hill, Inc., New York, NY, USA, 2009.

[55] Thomas Panas. Signature visualization of software binaries. In *SoftVis '08: Proceedings of the 4th ACM symposium on Software visualization*, pages 185–188, New York, NY, USA, 2008. ACM.

[56] Michalis Polychronakis, Panayiotis Mavrommatis, and Niels Provos. Ghost turns zombie: exploring the life cycle of web-based malware. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–8, Berkeley, CA, USA, 2008. USENIX Association.

[57] Paulraj Ponniah. *Data Warehousing Fundamentals*. John Wiley & Sons, Inc., New York, NY, USA, 2001. Foreword By-Reddy, Pratap P.

[58] Gavin Powell. *Beginning XML Databases (Wrox Beginning Guides)*. Wrox Press Ltd., Birmingham, UK, UK, 2006.

[59] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iframes point to us. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[60] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The ghost in the browser analysis of web-based malware. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[61] Daniel A. Quist and Lorie M. Liebrock. Visualizing Compiled Executables for Malware Analysis. `http://www.offensivecomputing.net/vizsec09/dquist-vizsec09.pdf`, 2009.

[62] Maurizio Rafanelli, editor. *Multidimensional databases: problems and solutions*. IGI Publishing, Hershey, PA, USA, 2003.

[63] Steve Riley. Exploring The Windows Firewall. `http://technet.microsoft.com/en-us/magazine/2007.06.vistafirewall.aspx`, 2007.

[64] Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy Zawodny, Arjen Lentz, and Derek J. Balling. *High Performance MySQL, 2nd edition*. O'Reilly, 2008.

[65] Craig Scott, Kofi Nyarko, Tanya Capers, and Jumoke Ladeji-Osias. Network intrusion visualization with niva, an intrusion detection visual and haptic analyzer. *Information Visualization*, 2(2):82–94, 2003.

[66] Brett Stone-Gross, Andy Moser, Christopher Kruegel, Engin Kirda, and Kevin Almeroth. FIRE: FInding Rogue nEtworks. In *25th Annual Computer Security Applications Conference (ACSAC)*, page 10, 2009.

[67] Philipp Trinius, Thorsten Holz, Jan Göbel, and Felix C. Freiling. Visual Analysis of Malware Behavior Using Treemaps and Thread Graphs. `http://pi1.informatik.uni-mannheim.de/filepool/publications/THGF_vizsec2009.pdf`, 2009.

[68] Edward Tufte. *Envisioning information*. Graphics Press, Cheshire, CT, USA, 1990.

[69] Edward R. Tufte. *The Visual Display of Quantitative Information, 2nd edition*. Graphics Press LLC, 2001.

[70] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[71] Kevin Williams and Michael Brundage. *Professional XML Databases*. Wrox Press Ltd., Birmingham, UK, UK, 2000.

[72] Xiaoxin Yin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju. Visflowconnect: netflow visualizations of link relationships for security situational awareness. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 26–34, New York, NY, USA, 2004. ACM.

[73] Klaus Zeuge, Troy Rollo, and Ben Mesander. The Client-To-Client Protocol (CTCP). `http://www.irchelp.org/irchelp/rfc/ctcpspec.html`, 1994.