PROCESS ACTIVITY LOGGERWITH ROOTKIT

By

NC Muhammad Umair

NC ZaghamMehmood

NC Saad Aziz

Submitted to the Faculty of Computer Software Engineering

National University of Sciences and Technology, Islamabad in partial fulfillment

for the requirements of a B.E. Degree in Computer Software Engineering

JUNE 2013

CERTIFICATE

Certified that the contents and form of project report entitled **"PROCESS ACTIVITY LOGGER WITH ROOTKIT"** submitted by 1) Muhammad Umair, 2) ZaghamMehmood, and 3) Saad Aziz have been found satisfactory for the requirement of the degree.

Supervisor: _____

Maj. Dr. Baber Aslam

DECLARATION

No portion of the work presented in this dissertation has been submitted in Support of another award or qualification either at this institution or elsewhere.

Dedication

In the name of Allah, the Most Merciful, the Most Beneficent.

This dissertation is dedicated to our parents and our supervisor Maj. Dr. Baber Aslam for their unwavering faith in us, their continuous support and love without which we would not have been able to succeed.

ACKNOWLEDGEMENTS

There is no success without the will of ALLAH. We are grateful to ALLAH, who has given us guidance, strength and enabled us to accomplish this task. Whatever we have achieved, we owe it to Him, in totality. We are also grateful to our parents and family and well-wishers for their admirable support. We would like to thank our supervisors Maj. Dr. Baber Aslam, for his help and motivation throughout the course of our project. Without hishelp we would have not been able to accomplish anything. He provided us with the opportunity to polish our technical skills and guided us into this area of learning.

TABLE OF CONTENTS

List of Figures

Figures **Page Number**

List of Tables

Table NumberPage Number

CHAPTER – 1

INTRODUCTION

1. Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of the Hidden Activity Logger with Rootkit. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli.

1.2 Project Scope

1.2.1 Short description of Project:

The project deals with two prominent and complimentary aspects of information security; One is monitoring and logging processes and activity on a system, whereas the other revolves around hiding the existence of processes and thus their activity on a similar system. Consequently, there are two distinct features to this project:

• An activity logger

- *Monitoring Processes and their activity*

• A Rootkit

- *Hiding the existence of activity logger itself*

1.2.2 Objectives:

1.2.2.1 Activity Logging Objectives

- Network Activity Monitoring
    - Connections initiated by a process
        - Source port

        - Destination IP/Port
        - Protocol
- File-System Monitoring
    - Files created, deleted or modified by a process
- Registry Monitoring
    - Registry ‘keys’ and ‘values’ added, deleted or modified by a process
- Keylogging
    - Keystroke recording, that are typed on the target system

1.2.2.2RootKit Objectives

- Not show up in task/process managers
- Not show up as a service
- Evade other common detection mechanisms e.g. process enumeration tools
- Provide stealth for the activity logger

1.2.3 Purpose:

Developing a RootKit and monitoring the activity of processes, network traffic etc requires understanding and skills at a very low level where the OS API works. This involves interception of System API calls, network connections and so on. To learn these immensely enhances knowledge of the operating system, and how its kernel works under the hood. This is the same kind of knowledge that hackers use to circumvent security in systems, whereas the ‘whitehats’ use to prevent that same circumvention. In today’s environment, there is a great emphasis on learning the higher level technologies, whereas kernel/lower level stuff leaves a void to be filled. This is the motivation behind developing a project,

where concepts such as rootkits and process activity interception and logging (low level fiddling) come into play. The purpose is to acquire these skills, while developing a product that is equally useful in the Information Security Stream.

1.2.4 Benefits:

- It can be used as a tool for behaviour analysis of processes, in order to detect for example malicious programs etc (programs that are not performing the activity that was advertised for instance)
- It can also be used for generic activity monitoring on any system

1.3 Objectives

The objectives of our project include:

- ✓ Develop a process activity logger that logs the following activities of the selected processes
    - o Network Activities
    - o File System Activities
    - o Registry Activities
    - o Keylogging
- ✓ Develop a rootkit that will allow the activity logger to stay hidden in the system
- ✓ Develop a GUI client application that will allow the user to remotely configure the activity logger and to request activity logs that have been generated on the target system.

1.4 Deliverables

Deliverable Name	**Deliverable Summary Description**
Software Requirements Specification(SRS) Document	Complete Description of WHAT system will do, who will use it. Detailed description of functional and non-functional requirements and system features.
Analysis Document	Detailed requirement analysis and analysis models are included.
Design Document	Complete description of How the system will do. Design models are included.
Code	Complete code with the API.
Testing Document	Whole system is tested corresponding to the specifications. System is tested at all levels of Software Development Life Cycle (SDLC).
Complete System	Complete working system.

CHAPTER – 2

LITERATURE REVIEW

2. Literature Review

Our project and thus the literature review comprises of two primary components:

- Rootkits
- Activity Logging

2.1 Problem Domain and Related Work

The concept of rootkits is a taboo one, and their primary purpose remains to provide stealth. They have been known to be used in viruses, worms and other forms of malwares, including the notorious “stuxnet” that was intended to sabotage Iran’s Nuclear Enrichment Facilities. However, rootkits have seen more mainstream uses as well e.g. Sony BMG published CDs with copy protection and digital rights management software called Extended Copy Protection. The software included a music player but silently installed a rootkit which limited the user's ability to access the CD.

On the other hand activity logging is as the name suggests recording certain types of activities on a particular system. In this case, the activities of certain processes running on a system are targeted to get recorded. Combining these two components can make for a powerful tool, a process activity logger with stealth, which has its fair share of applications. For instance, law enforcement agencies use such tools to aid their surveillance and investigations.

- FBI developed a program known as ‘Magic Lantern’ which according to reports could be installed via email or a software exploit. Once installed the program surreptitiously logged keystrokes.

- CIPAV (Computer and Internet Protocol Address Verifier) is a tool, also developed by FBI. When installed CIPAV funnels information about the targeted host (e.g. network configuration, running processes, IP connections) back to authorities.

2.2 Shortcomings/issues

The problem with rootkits is that the vendors of the operating systems that these rootkits run on, never intended for any activity or process on their system to hide itself. Instead, the conventional operating systems are designed to enumerate all the files, processes and services running on the system and make them visible to the user. Thus the techniques that rootkits employ to enable stealth tend to become obsolete as the vendors such as Microsoft release patches for the underlying mechanisms that allowed those techniques to work. However, new ideas/techniques to develop rootkits keep popping up every once in a while, some more effective than others.

Although many activity logging solutions including Trojans exist, but no product that can log the activity of processes that have been selected on the run time exists in the mainstream. To add to that is the element of stealth, which makes the product even more powerful. Therefore our project comprises of the development of a “Process Activity Logger”, for the Windows OS and to research, find and study working stealth techniques, which will allow for the development of a rootkit, which will enable the activity logger to function in the system surreptitiously.

Based on a report by NetMarketShare (http://netmarketshare.com), which concluded that Windows 7 is currently the most used Microsoft OS worldwide, we have chosen Windows 7 (32 bit) as our target platform for the rootkit and activity logger.

CHAPTER 3

SYSTEM REQUIREMENT SPECIFICATION

3. System Requirement Specification

3.1 Document Conventions

Most requirement statements in this section have equal priorities and importance, as failure of one of them would not lead to collapse of the whole system. This should be kept in mind that this document describes a stealth application. Thus the user interaction is minimal, and only required for configuration purposes. Therefore most of the functions are automated by their nature and minimal use case representation is present.

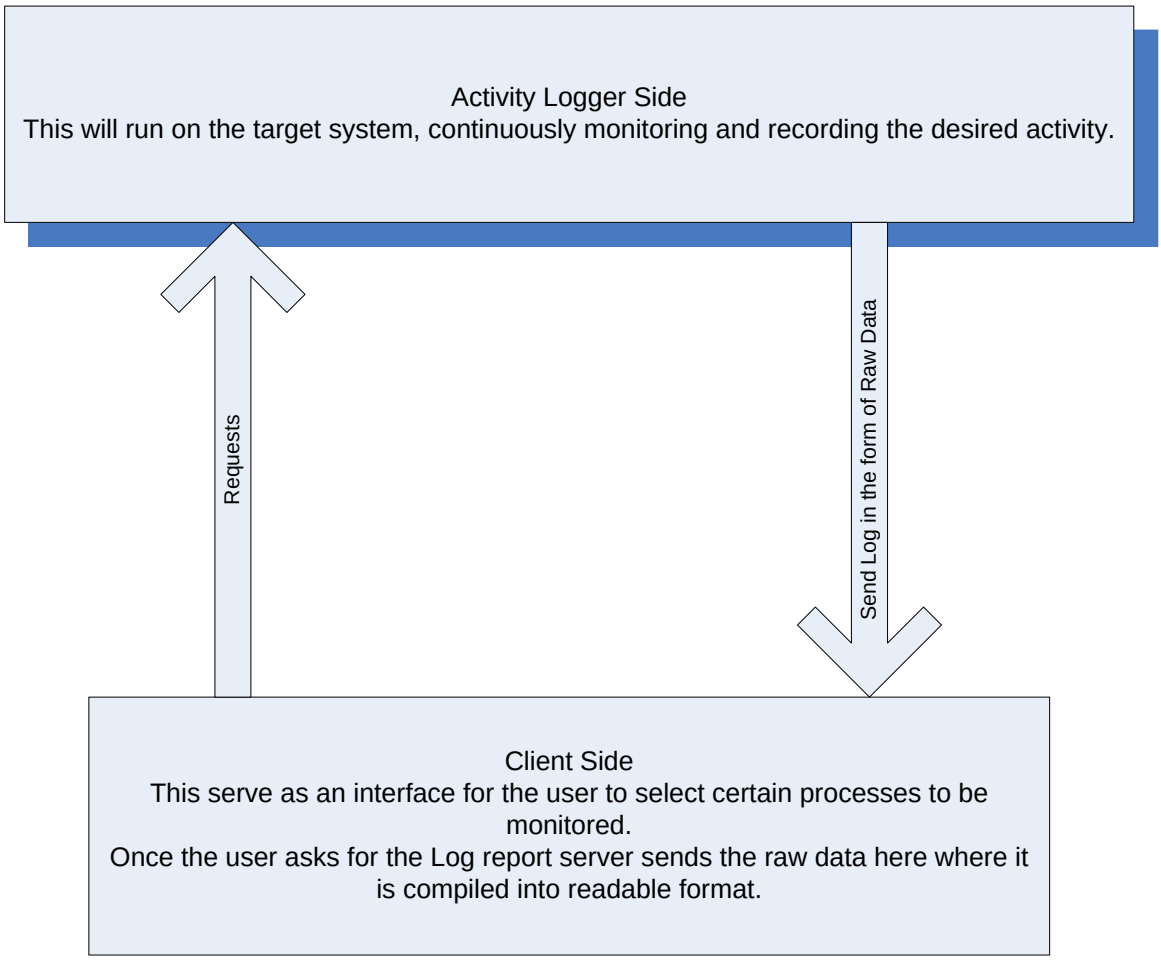
3.2 Overall Description

3.2.1 Product Perspective

This Activity Logger and Rootkit combo is a new self-contained product. This can make for a powerful combination that has its fair share of applications. For instance, law enforcement agencies use such combinations to aid their surveillance and investigations.

- FBI developed a program known as ‘Magic Lantern’ which according to reports could be installed via email or software exploit. Once installed the program surreptitiously logged keystrokes.
- CIPAV (Computer and Internet Protocol Address Verifier) is a tool, also developed by FBI. When installed CIPAV funnels information about the targeted

host (e.g. Network configuration, running processes, IP connections) back to authorities.



3.2.2 Product Functions

The major functions that will be provided by the product are:

- User will be able to monitor the desired process/processes on target systems.

- User can monitor the activities over the network of a single or multiple target systems
- Each Activity will be logged in raw form on the target computer.
- Users can specify a single process to be monitored or multiple processes, from the client GUI application
- User will be able to monitor if files are being written, renamed, deleted or modified on the target system depending on the process/processes being monitored.
- User will be able to monitor the connections initiated by a process.
- User will be able to monitor the registry i.e. any change in registry IDs and values will get logged
- User will also be able to monitor the keystrokes made on the target system
- The target computer whose activity is to be monitored sends the data in raw format to the client computer where it is compiled and summarized into a readable format.

3.2.3 User Classes and Characteristics

The system is being developed for different classes of end users. End users can be Security Analysts, Malware analysts, Intelligence Agencies, Law Enforcements agencies etc.

- **Intelligence/Law Enforcement Agencies**

  They can use this tool in their surveillance operations like was previously and notoriously used by the FBI.

- **Security/Malware Analysts**

  Security and Malware Analysts can use this tool to analyze potentially malicious files and their behavior, even remotely. They will be able to determine the nature of a process or file by the type of activity it has performed, and the type of activity it was supposedly performing.

3.2.4 Operating Environment

The System will be deployed on Intel Processor architectures, running Microsoft windows 7 (32-bit version). The activity logger with the RootKit will be run on a target computer. This part of the software can be referred to as the server. The client side is where the administrator configures the rootkit and activity logger, and where logs are requested from target systems and summarized and compiled into readable formats.

The product will not stop working if an antivirus is already running on the system nor will it interrupt other processes. It will only sit surreptitiously, intercepting and logging activity, without itself interfering in any operations or making itself known.

3.2.5 Design and Implementation Constraints

The system will be designed and developed under the Following constraints:

- If any employees, or other relevant people are going be using systems that are under monitoring by the Activity Logger, it is required by the

company to have legally signed contracts by those people which allows it to gather such information from under their usage which might contain personal data.

- As the system is targeted for Windows 7 (32-bit), it may not function properly on other versions of Windows. The reason for this choice is that Windows 7 is currently the most used Microsoft Operating System worldwide, as concluded by a report by NetMarketShare (http://netmarketshare.com).
- No matter the intent, or the usage type, the activity logger has to remain hidden on the target system.

3.2.6 User Documentation

Following user documents shall be provided with the system on deployment:

- A short quick-deployment guide
- A CD containing all the information about the system
- User Manual

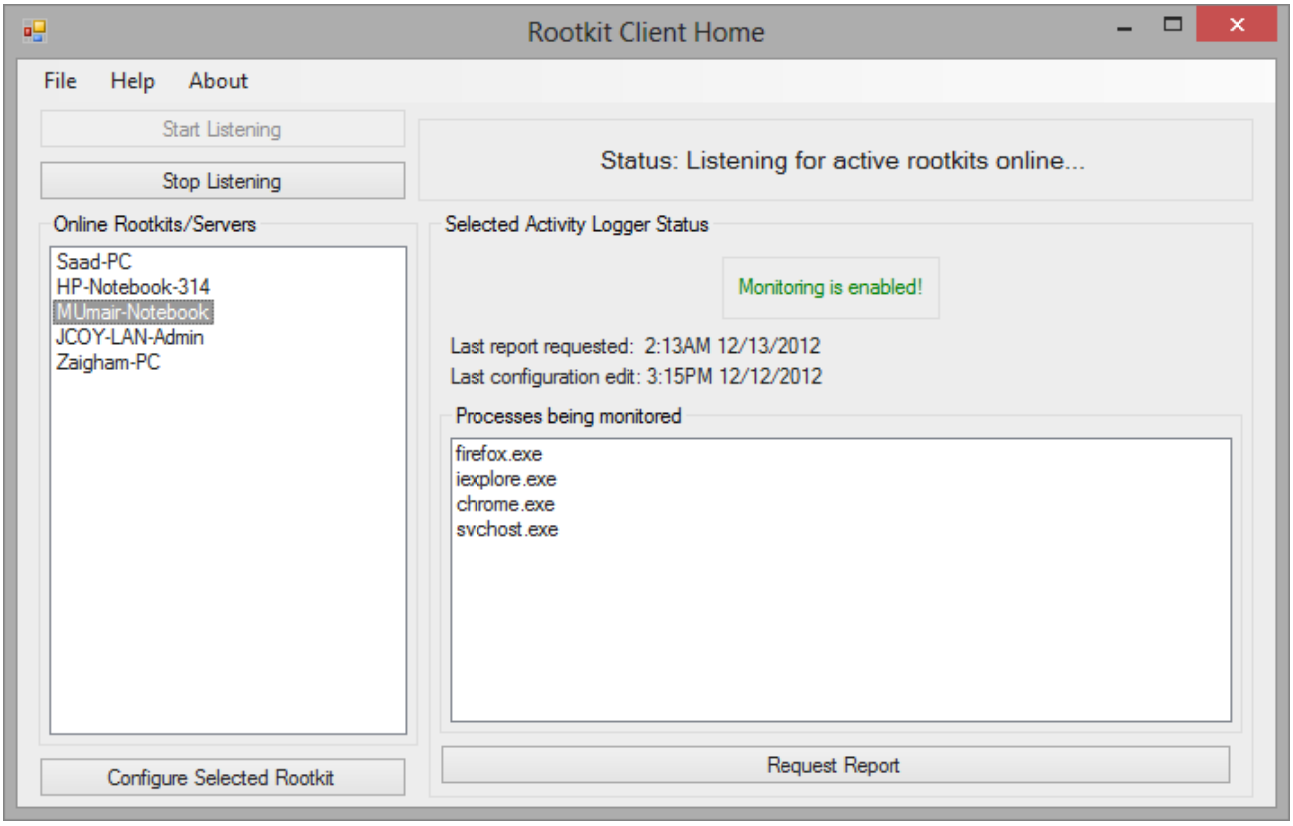
3.2.7 Assumptions and Dependencies

The key assumption is that, the user will have administrator level access to the system that he wants to be monitored. This although may not be necessary but might be required for the initial installation of the rootkit and activity logger. From there on, the rootkit will handle the privilege level itself locking on to the kernel mode or ring-0, regardless of the privilege level of the logged-in user.

3.3 External Interface Requirements

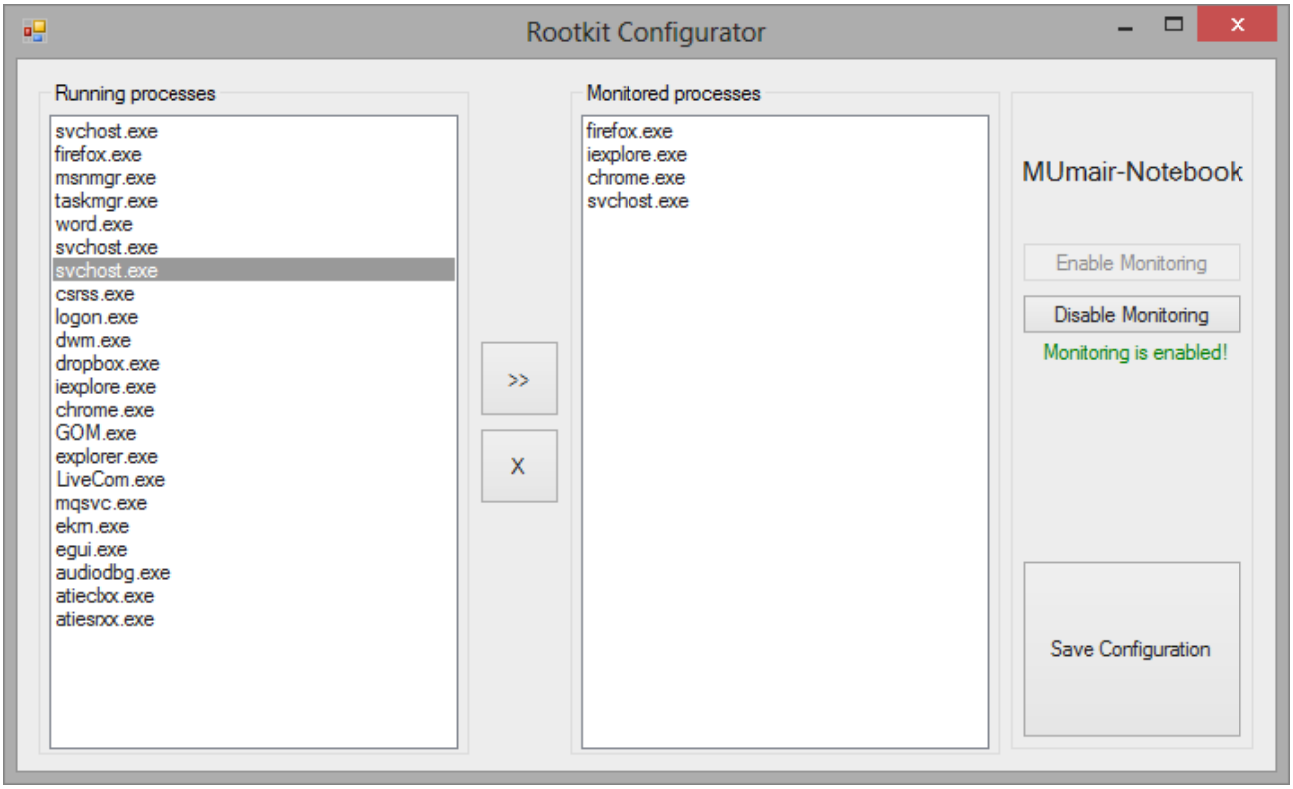
3.3.1 User Interfaces

The rootkit/activity logger part of the product does not have a user interface as its main concern apart from other previously discussed points is stealth. Most of the tasks are automated and require minimal configuration that can be done remotely via the Rootkit Client application. Second purpose of the client is to receive logs from the activity logger, and parse them into summarized and readable formats. Below is a crude concept of what the activity logger client application would look like:



This conceptualizes the home screen of the Rootkit Client. Here we can see the rootkits that are active and available online. Upon selecting an available rootkit a summary of the configuration and last retrieved report is displayed; the summary includes the processes that are currently being monitored on the selected target system. Furthermore, an option to retrieve the log report has also been presented.

Another button takes us into the Rootkit Configurator screen:



Here the monitoring can be turned on or off. Furthermore, all the processes that are currently running on the remote system are displayed. Any of these processes can be set to be monitored by the push of a button. Lastly, the configuration can be saved and updated to the activity logger on the target system.

3.3.2 Hardware Interface

The rootkit can be deployed on home computers, laptops, network computers or even servers as long as they are running the Window 7 32-bit Operating System. The client can be run on similar hardware.

3.3.3 Software Interfaces

3.3.3.1 Operating System

3.3.3.1.1 For End User

Microsoft Windows Operating Systems supporting .Net Framework 4.0 and above.

3.3.3.1.2 For System

The rootkit itself requires a Microsoft Windows and 32-bit operating system. That is the only prerequisite.

3.3.3.2 Programming Interface

- Microsoft Visual Studio (for rootkit development in C/C++)
- Microsoft Windows API
- Microsoft Windows Driver Kit (WDK) (for filter driver development)
- Any commonly available assembly compiler (might be needed during rootkit development)
- Kernel Debug Tools such as the KD (to debug filter drivers)
- .Net Development Framework (for client development)

3.3.4 Communications Interfaces

The primary over-the-net communication will occur between the rootkit/activity logger and the user application that is used for its configuration. The user application is also where the rootkit will transfer its generated log files to. In this scenario, the rootkit/activity logger will be acting as a server, and the GUI application on the user end will be acting as the client.

The method of communication will primarily be TCP/IP Sockets.

3.4 System Features

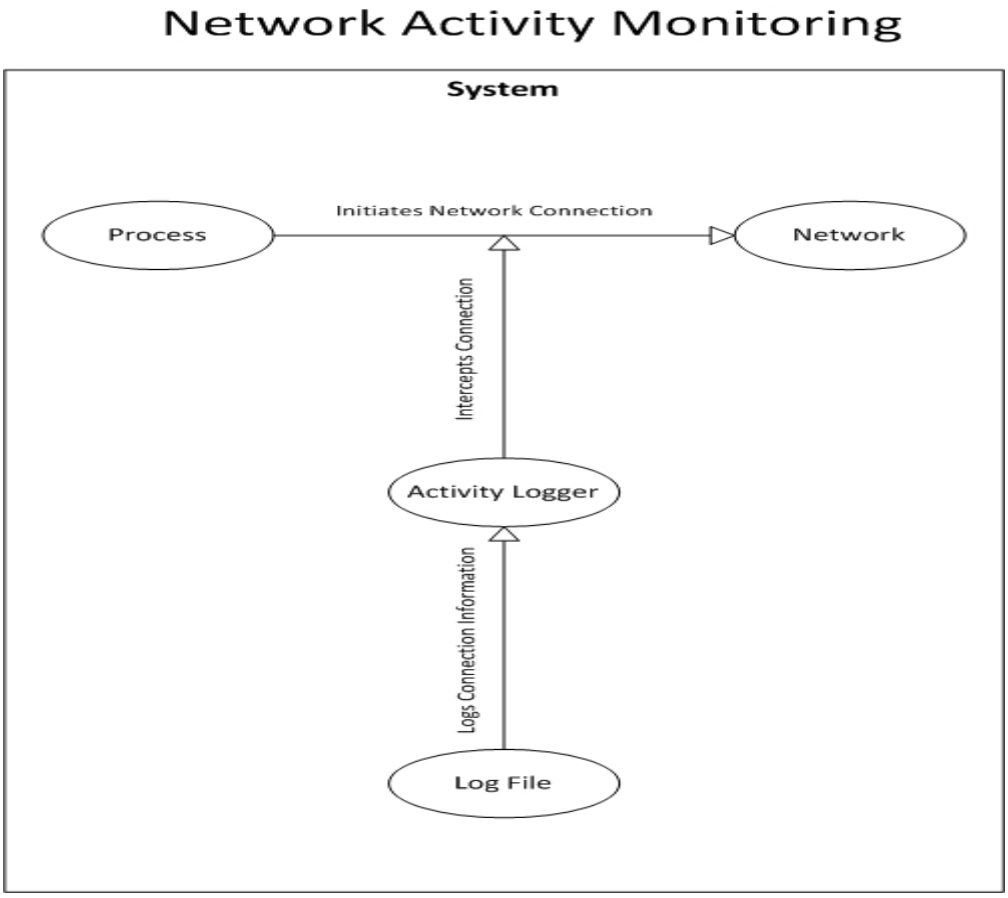
3.4.1 Network Activity Monitoring

3.4.1.1 *Description and Priority*

Whenever a connection is initiated by a monitored process on the target computer, for data transfer or any other reason, it gets recorded by the activity logger. The following components refer to each connection that gets logged:

- Source Port
- Destination IP/Port
- Protocol

There are no priorities in these components. This feature is selected by default when the user configures the rootkit to monitor any process/processes and thus will receive equal priority as any other feature.



3.4.1.2 *Stimulus/Response Sequences*

- The activity logger is installed on the system
- It connects to the client/configurator
- The user configures the rootkit/activity logger from the client and selects process/processes to be monitored
- The network activity of the selected processes then starts being recorded/logged.

3.4.1.3 *Functional Requirements*

The functional requirements for this feature are:

- A Network Monitoring Module will be inserted into the TCP/IP Stack. This module will be used to intercept the inbound and outbound network traffic and extract the previously mentioned information for purposes of log generation.

3.4.2 File-System Monitoring:

3.4.2.1 *Description and Priority*

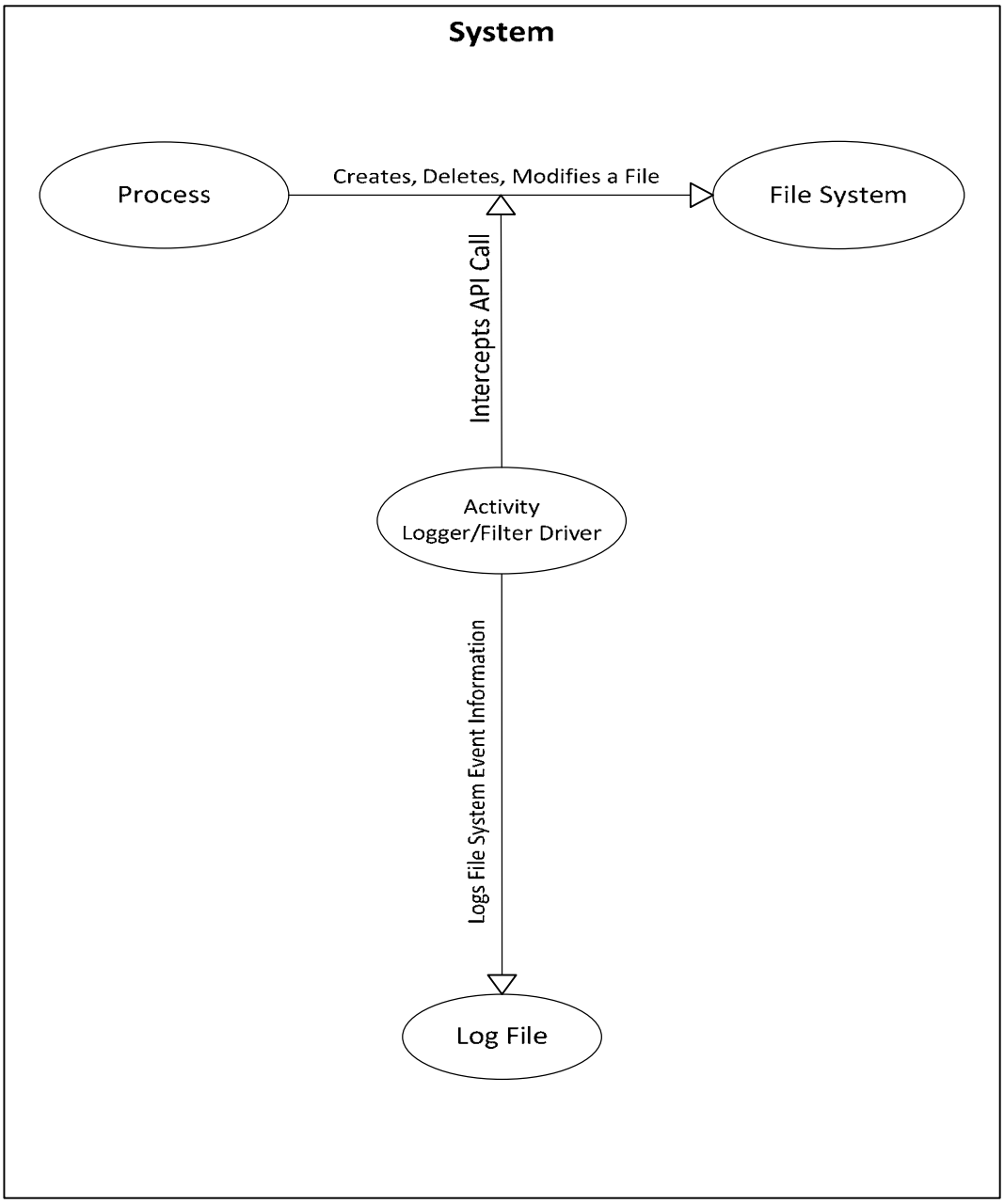
Whenever an activity is performed on the file system by a monitored process on a target system, it gets logged by the activity logger. For example, whenever a file is created, deleted or modified, renamed etc, it will get recorded in the logs that are being generated.

As mentioned earlier, there are no priorities in these components. This feature is selected by default when the user configures the rootkit to monitor any process/processes and thus will receive equal priority as any other feature.

3.4.2.2 *Stimulus/Response Sequences*

- The activity logger is installed on the system
- It connects to the client/configurator
- The user configures the rootkit/activity logger from the client and selects process/processes to be monitored
- The file system activity of the selected processes then starts being recorded/logged

File System Monitoring



3.4.2.3 *Functional Requirements*

The functional requirements for this subsystem are:

- A 'filter driver' will be working in the kernel-mode of the system. It will sit between the processes and the Operating System/Kernel and intercept any API calls made by the processes. A list of API calls pertaining to file system activity will be hardcoded into the activity logger. Whenever a call included in that list will be made by a process, the activity logger will make an entry of that call in the logs.

3.4.3 Registry Monitoring

3.4.3.1 *Description and Priority*

Whenever a process that is set to be monitored by the activity logger in a target system performs any operation on the system registry, it will get recorded in the logs that are being generated. This registry activity basically refers to the addition, removal or modification of the 'IDs' and their corresponding 'values' in the registry.

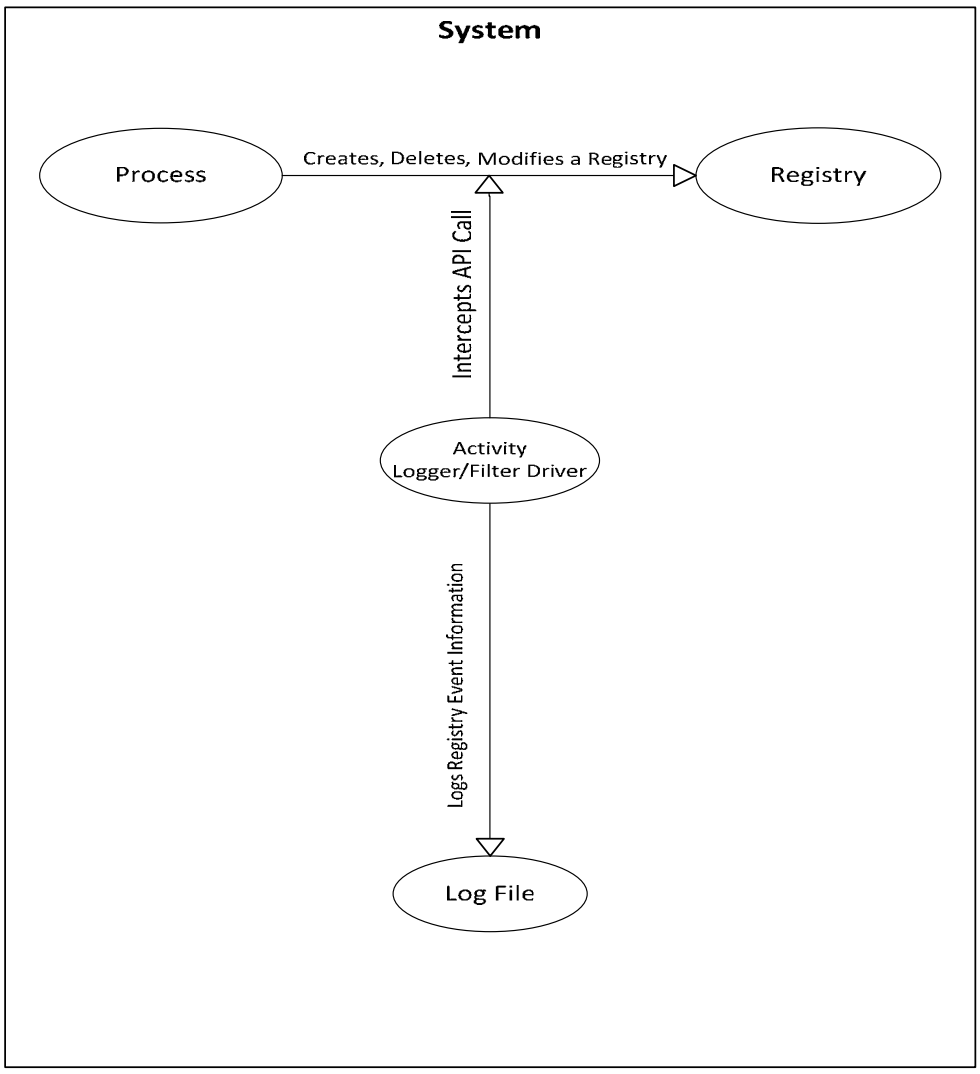
As mentioned earlier, there are no priorities in these components. This feature is selected by default when the user configures the rootkit to monitor any process/processes and thus will receive equal priority as any other feature.

3.4.3.2 *Stimulus/Response Sequences*

- The activity logger is installed on the system
- It connects to the client/configurator

- The user configures the rootkit/activity logger from the client and selects process/processes to be monitored
- The registry activity of the selected processes then starts being recorded/logged.

Registry Monitoring



3.4.3.3 *Functional Requirements*

Functional requirements for this subsystem are:

- A 'filter driver' will be working in the kernel-mode of the system. It will sit between the processes and the Operating System/Kernel and intercept any API calls made by the processes. A list of API calls pertaining to operations on the operating system registry will be hardcoded into the activity logger. Whenever a call included in that list will be made by a process, the activity logger will make an entry of that call in the logs.

3.4.4 Keystroke Logging

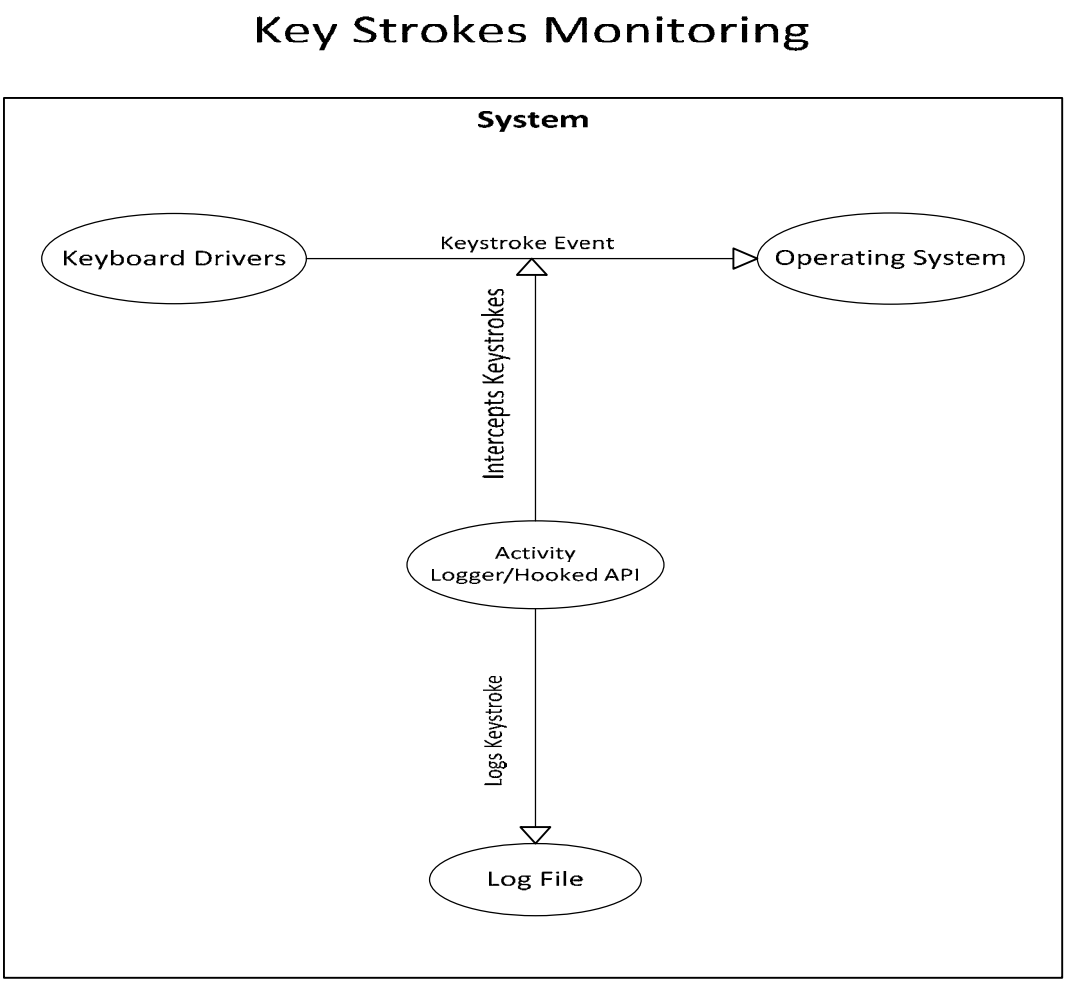
3.4.4.1 *Description and Priority*

The keys typed on the target system that is being monitored will get logged by the activity logger.

As mentioned earlier, there are no priorities in these components. This feature is selected by default when the user configures the rootkit to monitor any process/processes and thus will receive equal priority as any other feature.

3.4.4.2 *Stimulus/Response Sequences*

- The activity logger is installed on the system
- It connects to the client/configurator
- The user enables monitoring on the activity logger
- The keystrokes start being recorded in a log file on the target system.



3.4.4.3 *Functional Requirements*

Functional requirements for this subsystem are:

- The API Calls of the hardware driver of the keyboard, pertaining to keystrokes will be hooked into to allow for interception of the keys that have been typed so they can get logged.

3.4.5 Activity Logger configuration

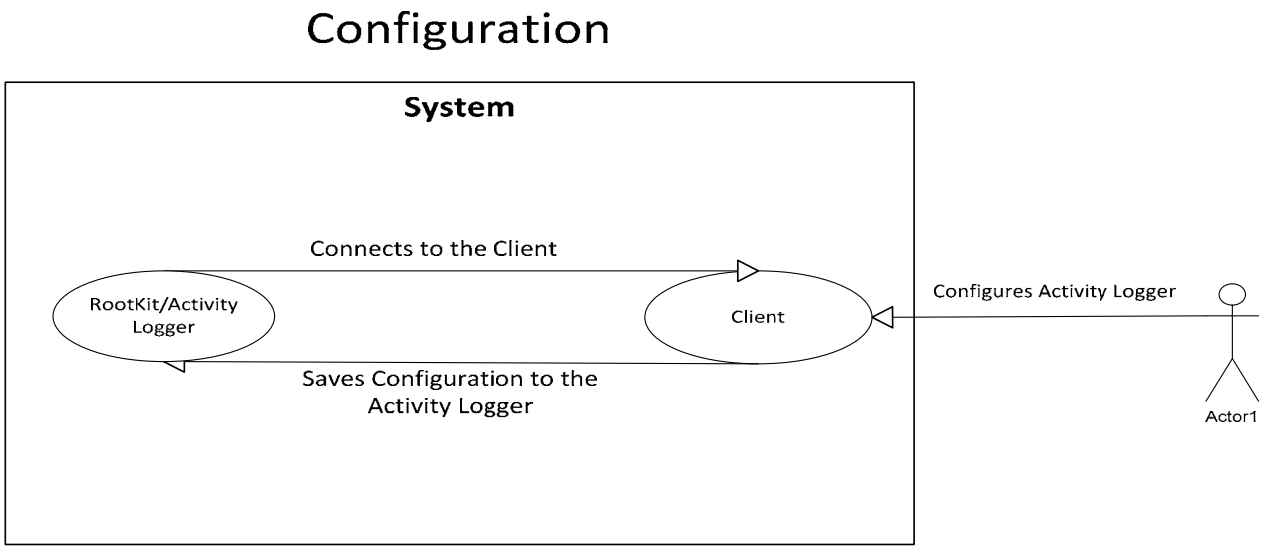
3.4.5.1 *Description and Priority*

The rootkit/activity logger will connect to the client application via a reverse connection. This will allow the client to remotely configure the activity logger, specifying which processes would get monitored and so on.

This is a high priority feature, as the other features of the system depend on the correct configuration of the activity logger.

3.4.5.2 *Stimulus/Response Sequences*

- The rootkit/activity logger gets installed on the target system
- It initiates a connection to the client application
- The client application accepts the connection
- The user is then provided by a list of settings, the primary of which will be the list of processes that can be monitored
- The user will then select the processes he wants to get monitored and save the settings
- The activity logger will start monitoring the activity of those processes and logging it.



3.4.5.3 *Functional Requirements*

Functional requirements for this subsystem are:

- A TCP/IP sockets based client server architecture that will allow the rootkit(server) to initiate a reverse connection with the client, and send/receive data over the network that will be used for the configuration purposes of the rootkit
- A flexible configuration mechanism in the rootkit/activity logger that allows it to monitor processes that have been specified to it on the runtime.
- A configuration saving mechanism that allows the rootkit to retain its settings depending on the context

3.4.6 Log Transfer to the client

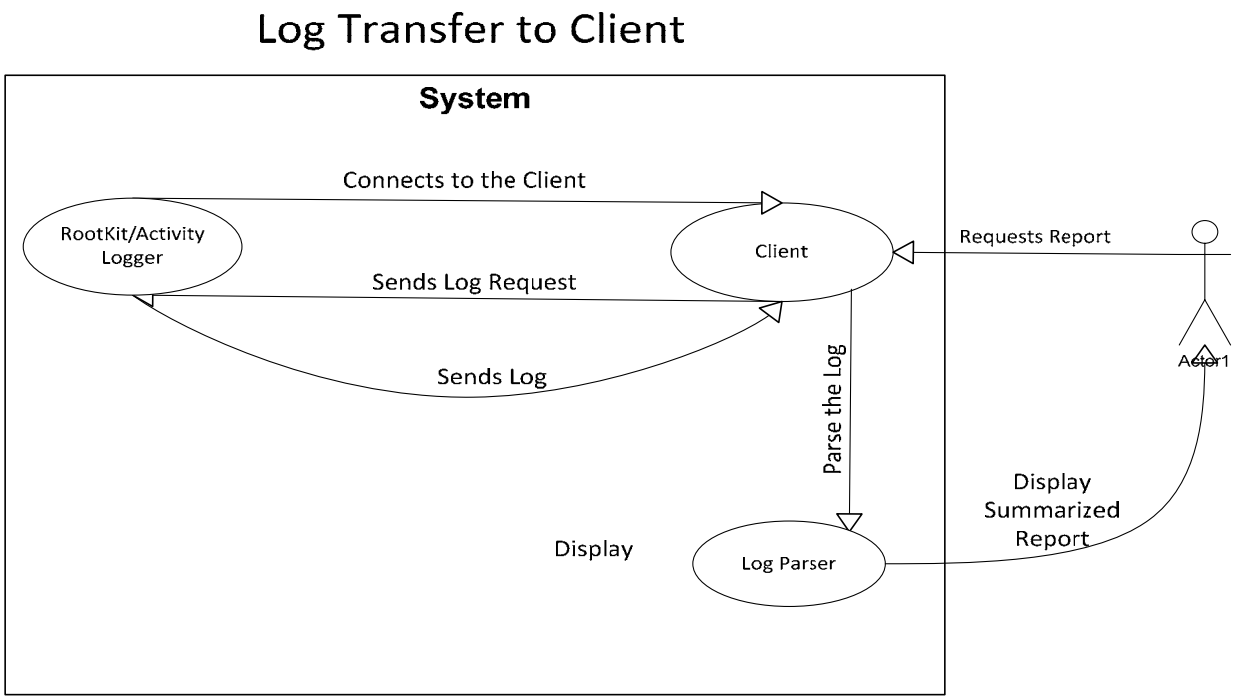
3.4.6.1.1 Description and Priority

The activity logger will transfer its generated logs to the client when requested.

This is a high priority feature, as if this fails the purpose of log generation would be killed since the user would not be able to receive the intended activity data.

3.4.6.2 *Stimulus/Response Sequences*

- The rootkit/activity logger is installed on the target system
- The rootkit initiates a reverse connection with the client
- The client accept the connection and the user is presented with the option to request generated logs (if any)
- The user requests the logs
- The logs are sent to the client, summarized, compiled and then displayed to the user



3.4.6.3 *Functional Requirements*

Functional requirements for this subsystem are:

- A TCP/IP sockets based client server architecture that will allow the rootkit(server) to initiate a reverse connection with the client, and send/receive data over the network that will include the raw logs generated by the activity logger
- A parsing and compiling mechanism on the client side, that will parse the raw logs, and generate a summary of the activity and compile it in a readable format for the user

3.4.7 Rootkit Functionality

3.4.7.1 *Description and Priority*

One of the primary concerns of the product is to work in stealth on the target systems. This is a feature that is extremely complex to implement and requires the use of various low level techniques. The rootkit terminology encompasses the whole of this concept.

This is a high priority feature, because if this fails the activity logger will lose its stealth capabilities.

3.4.7.2 *Stimulus/Response Sequences*

No particular stimulus or response is associated with this, as this is an "under-the-hood" feature. Its primary goal is to keep all of the activity logger components completely hidden in the system.

3.4.7.3 *Functional Requirements*

Functional requirements for this subsystem are:

- To hide the activity logger components from the task managers (native or third party)
- To hide the activity logger components from the process managers (native or third party)
- To hide the activity logger components from other file enumeration tools and scanners
- To hide the activity logger components from OS services enumeration
- To retain the stealth functionality even after system reboots by maintaining privileged access via operation in kernel-mode/ring-0

3.5 Other Non-Functional Requirements

3.5.1 Performance Requirements

Performance requirements are given as follow:

- They response to a request from client system to the rootkit takes not more than 10 seconds.
- The system's performance is not affected by the use of activity logger on the user computer.

- As soon as the data is transferred over to client it takes not more than 60 seconds to convert the raw data into readable format.
- The activity logger executable file is installed simply by double clicking, and does not require any hassle of configuration or location specification on the server end. The installation itself is instantaneous.

3.5.2 Security Requirements

The rootkit and the activity logger cannot be accessed or configured on the system that they are installed on. Not even a user with administrator privileges can access the rootkit. The activity logger can only be configured from the corresponding client application.

3.5.3 Software Quality Attributes

a. ***Correctness***

The system produces accurate activity results as performed by the process(es) during a certain period of time. These activities have been listed and explained in detail in prior sections.

b. ***Flexible***

Since software is flexible in nature, thus the system can incorporate new functionalities depending upon the companies requirements.

c. ***Maintainability***

After the release of the system, the system will be maintained after each patch is released by the Microsoft for window 7 to detect RootKit.

d. **Availability**

The activity logger will run 24/7 on the target computer without interfering with the system. Administrator/Client can log in any time and can request a report.

e. **Robustness**

If a system crash or failure occurs, the activity logger will resume its functionality once the system has been restarted. The logs will not be lost by the crash of the system.

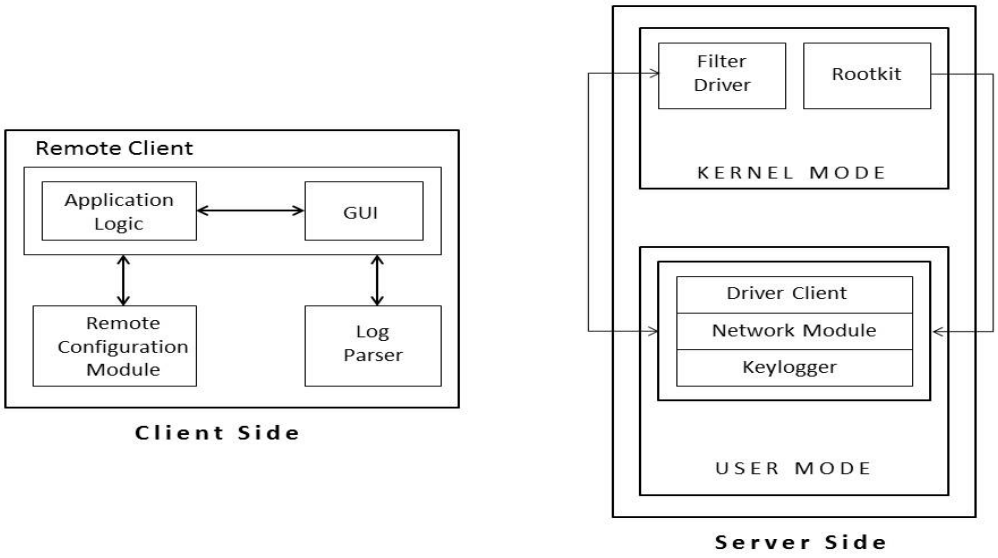
CHAPTER 4

SYSTEM DESIGN SPECIFICATION

4. System Design Specification

4.1 Architecture

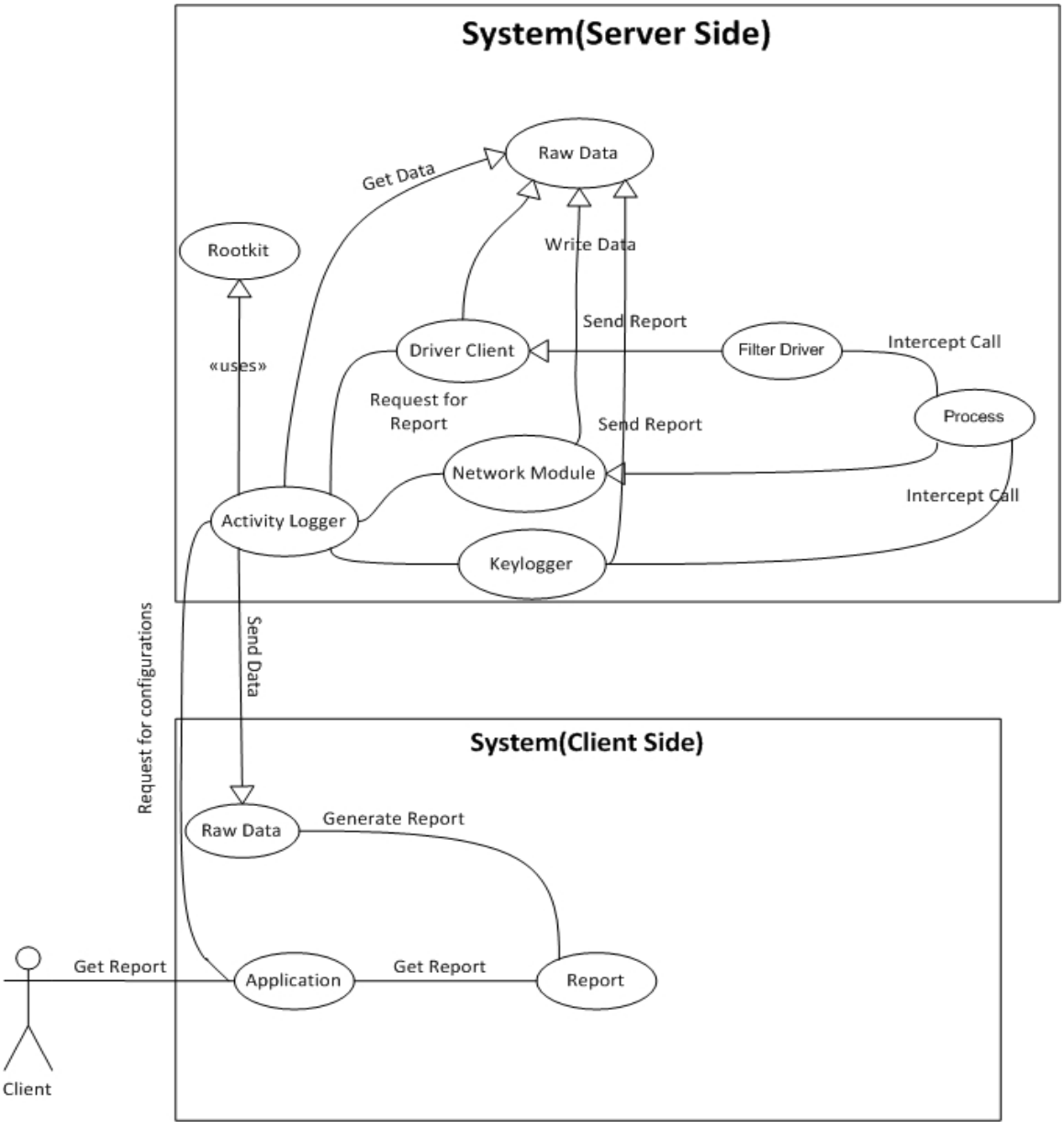
The product uses a client/server architecture to implement the stated functionality.



At the server side filter driver and rootkit work at the kernel level. Rootkit enables necessary components at the server side to remain hidden. The function of the filter driver is to intercept the filesystem I/O calls and registry I/O call made by the selected processes and logs them down in separate files for each process. Similarly the Network Monitoring Module monitors the activity taking place on the network (only activity of selected processes); this activity is reported to the Network Monitoring Module client, which logs it down. The keylogger records the keystrokes made on the system. At the client side the user/administrator sends request to the server side to fetch the data. The data is fetched in raw format and is parsed at the client side.

4.2 Detailed Design

4.2.1 System Use Case Diagram



The above diagram represents a summarized scenario of the system where activity logs are generated and then parsed in a readable summarized format.

This is a brief explanation of the system use case.

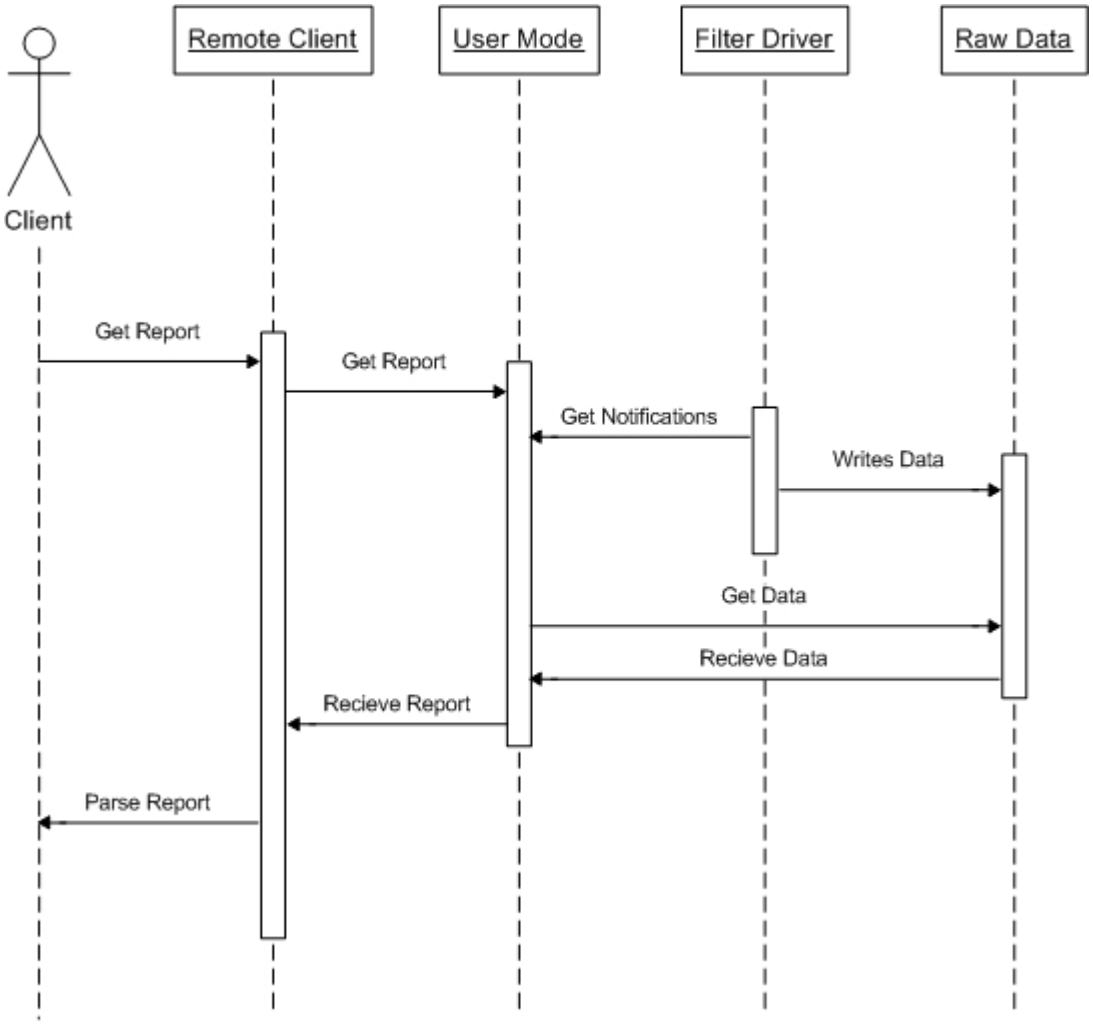
The logs are being maintained by the user mode components of the server side, with help from their corresponding kernel mode components. The user mode components include filter driver client, Network Monitoring Module client, Network Monitoring Module itself and the Key logger. The Kernel mode includes the filter driver itself. Additionally, the rootkit itself is running with kernel mode access to provide stealth to the necessary server side components.

- A report is requested by the client side
- The server sends a raw report to the client
- The raw data is parsed into readable format.

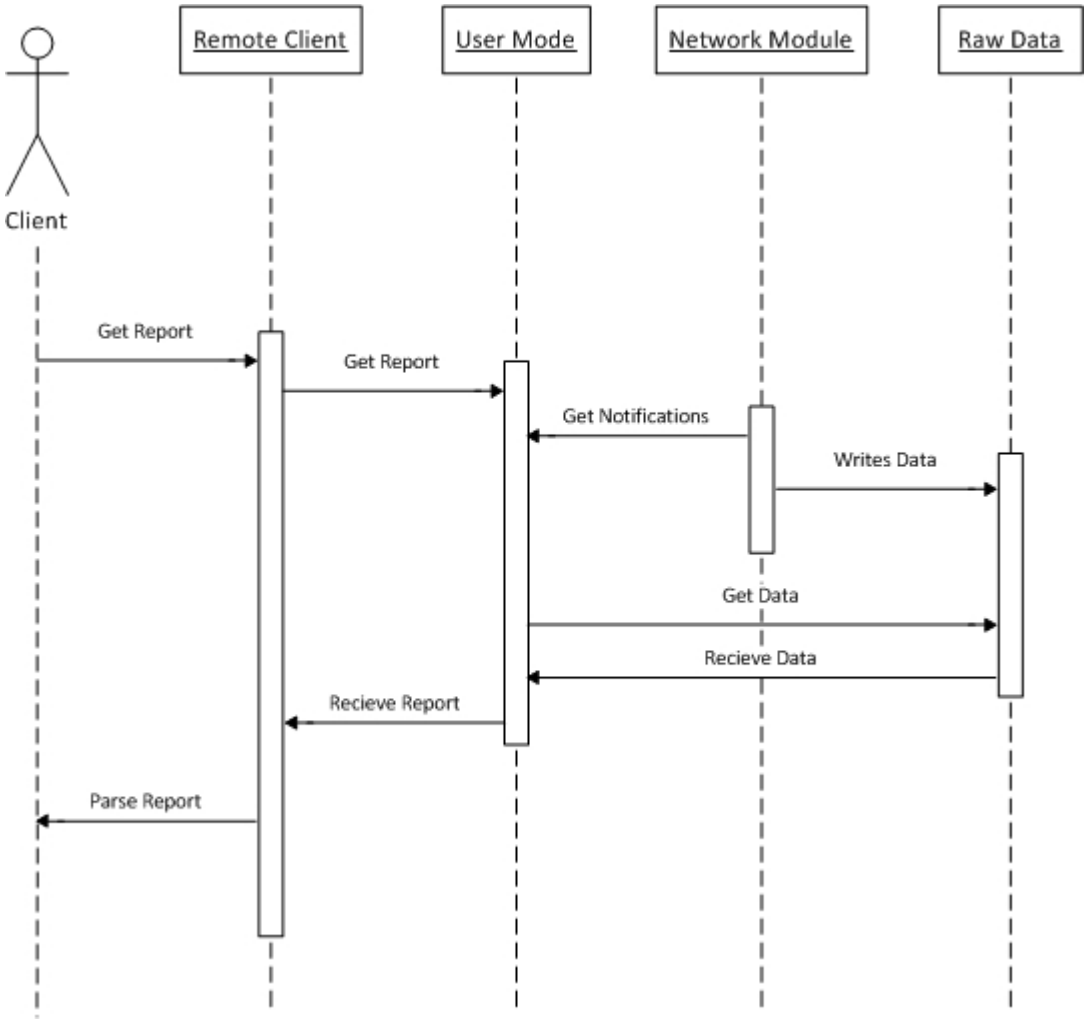
4.2.2 Logical View

4.2.2.1 *Sequence Diagram(Filter Driver)*

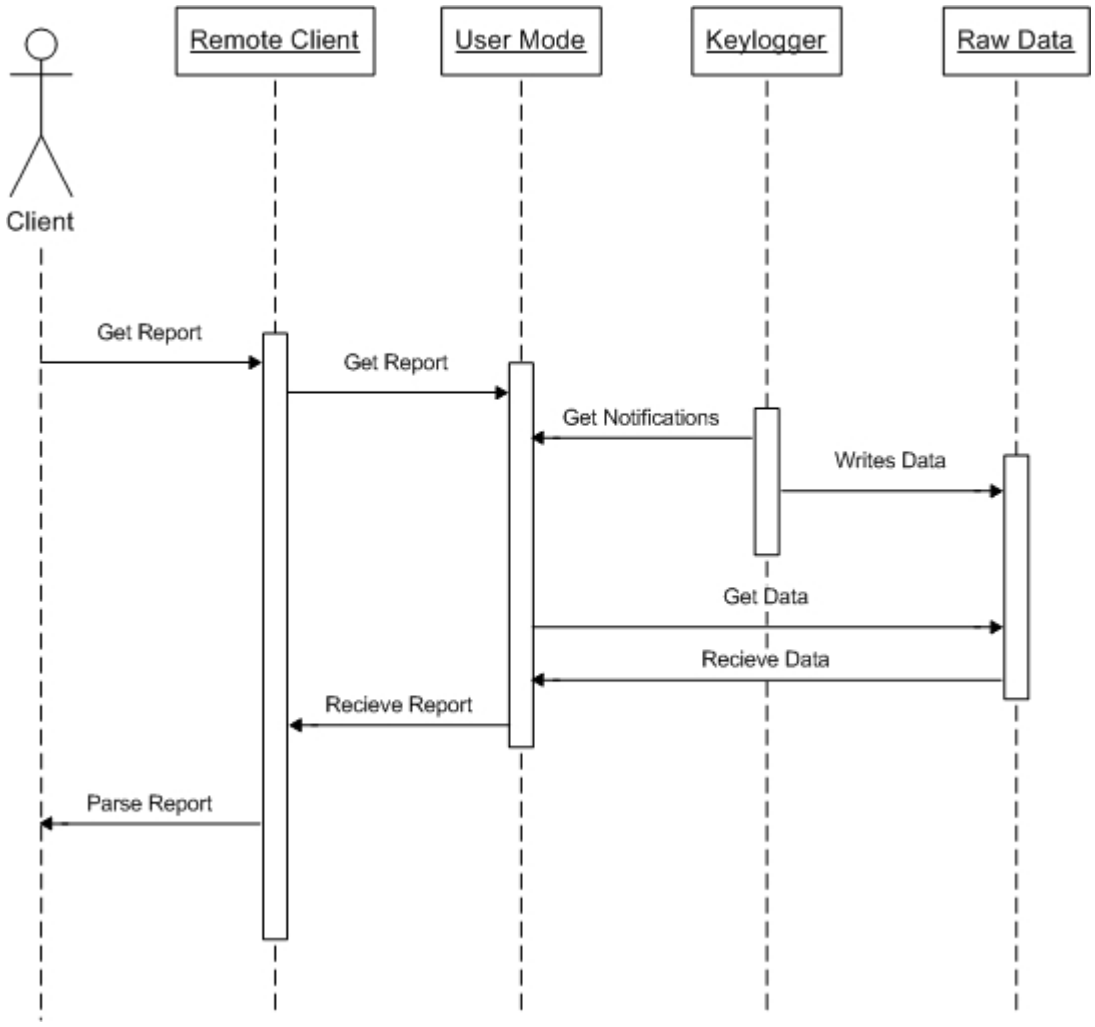
The three sequence diagrams present below describe the 3 states of the system.
In each state client is fetching the data from the server. This data is received via user mode through driver client, Network Monitoring Module client and Keylogger. The rootkit keeps necessary components at the server side hidden.


Remote Client
User Mode
Filter Driver
Raw Data
Client
Get Report
Get Report
Get Notifications
Writes Data
Get Data
Recieve Data
Recieve Report
Parse Report

4.2.2.2 *Sequence Diagram(Network Monitoring Module Client)*



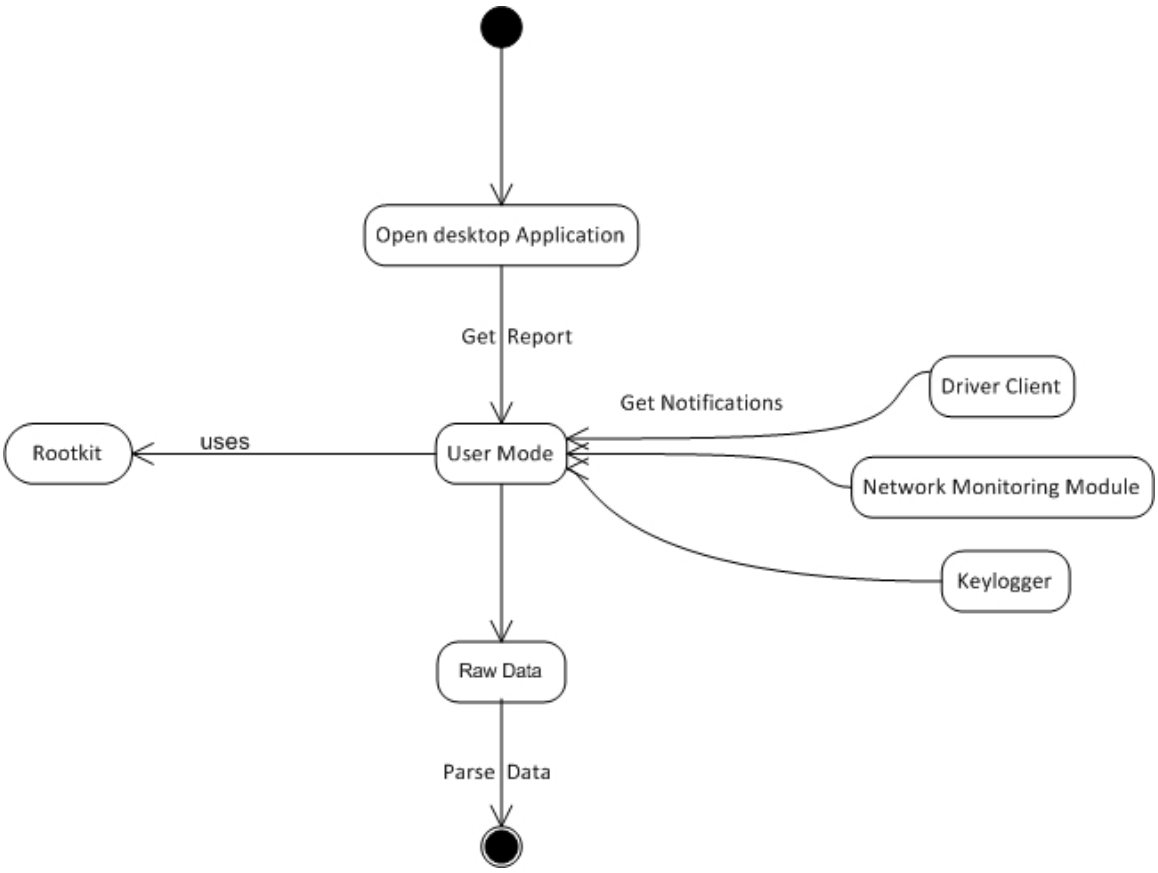
4.2.2.3 *Sequence Diagram(Keylogger)*



4.2.3 Dynamic View

4.2.3.1 *Activity Diagram*

The following diagram represents the generic activity flow of the application at the back end when a user requests for a report.



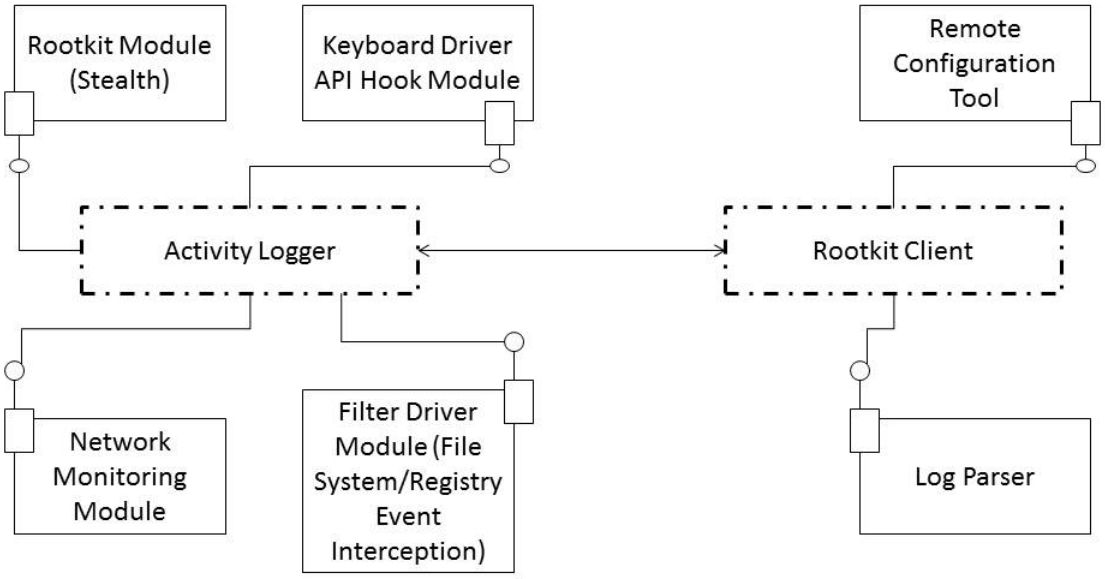
4.2.4 Implementation View

4.2.4.1 *Component Diagram*

Components involved in the system are:

- Activity Logger
- Rootkit Module
- Keyboard Driver
- Network Monitoring Module
- Filter Driver Module
- Rootkit Client
- Remote Configuration Tool
- Log Parser

The Keyboard API Hook Module is used to perform the keylogging. The Filter Driver and Network Monitoring Module perform the file system, registry and network activity monitoring and logging. Finally the Rootkit module provides stealth to the activity logger. On the client side, the remote configuration module is used to remotely configure the activity logger to monitor the specified processes. Whereas the log parser module, parses and generates summarized log reports from the raw data that has been received from the activity logger.


Rootkit Module (Stealth)
Keyboard Driver API Hook Module
Remote Configuration Tool
Activity Logger
Rootkit Client
Network Monitoring Module
Filter Driver Module (File System/Registry Event Interception)
Log Parser

Chapter 5:

SYSTEM IMPLEMENTATION

5. Implementation

The project comprises of the following modules/functionalities:

- Activity Logger
    - Rootkit
    - Registry activity monitor
    - File System activity monitor
    - Network activity monitor
    - Keystroke logger
    - Remote connection module
    - Process Name Retrieval
- Remote Client
    - Remote configuration module
    - Log Parser
    - Remote connection module

All these modules have been implemented in more than 5000 lines of code. Most of the code has been written in C/C++. While the frontend has been developed using .Net (C#) and Agility Pack. Implementation details module by module are as follows:

5.1 Rootkit

The rootkit employs the method of Direct Kernel Object Manipulation to hide the running process of the activity logger. The kernel uses certain structures to keep track of

the happenings and various states within the operating system. Gaining access to these structures and modifiying them to our gain is thus refered to as DKOM.

For our purposes in order to hide the running process we must first take a look at the the EPROCESS structure, that is used to represent a process in the kernel. The kernel maintains a doubly linked list of all the EPROCESS structures, in order to keep track of all the running processes (EPROCESS is an opaque/undocumented structure, but can be viewed via WinDbg). The main goal is to somehow gain access to this linked list, and find our desired process and then remove it from that list, so that it can run hidden in the system. To achieve this we require things:

1. Direct access to the linked list
2. Exclusive modification access/privlege so that a race condition does not occur

For the first goal, firstly we use the PsGetCurrentProcess() function, to acquire a pointer to the EPROCESS structure of the currently running process. The EPROCESS structure contains multiple fields, two of which are relevant for our purposes:

- UniqueProcessId (Pointer to a 32 bit value)
- ActiveProcessLinks (Structure of type LIST_ENTRY)

The UniqueProcessId as obvious, gives the 32 bit process identifier of the current EPROCESS.

A **LIST_ENTRY** structure describes an entry in a doubly linked list or serves as the header for such a list.

Syntax

C++

```
typedef struct _LIST_ENTRY {
  struct _LIST_ENTRY  *Flink;
  struct _LIST_ENTRY  *Blink;
} LIST_ENTRY, *PLIST_ENTRY;
```

Members

Flink

For a **LIST_ENTRY** structure that serves as a list entry, the **Flink** member points to the next entry in the list or to the list header if there is no next entry in the list.

For a **LIST_ENTRY** structure that serves as the list header, the **Flink** member points to the first entry in the list or to the LIST_ENTRY structure itself if the list is empty.

Blink

For a **LIST_ENTRY** structure that serves as a list entry, the **Blink** member points to the previous entry in the list or to the list header if there is no previous entry in the list.

For a **LIST_ENTRY** structure that serves as the list header, the **Blink** member points to the last entry in the list or to the **LIST_ENTRY** structure itself if the list is empty.

In this case, the Flink points to the next process'es EPROCESS structure, where the Blink points to the previous EPROCESS structure. Now to get to this field we must add the offset of the UniqueProcessId, into the pointer of the EPROCESS stucture (PEPROCESS), as shown below:

LEntry = *((LIST_ENTRY*)(currEP + LE_OFFSET));

The LE_OFFSET in this case is: 0x0B8

Now we have access to the linked list, and access traverse it using it Flink and Blink fields, by adding their respective offsets into the pointer. For each EPROCESS structure we must look to its UniqueProcessId, to match whether this is the one representing the process that we wish to hide. This is done by subracting the offset of the EPROCESS, from the pointer of the LIST_ENTRY as shown below:

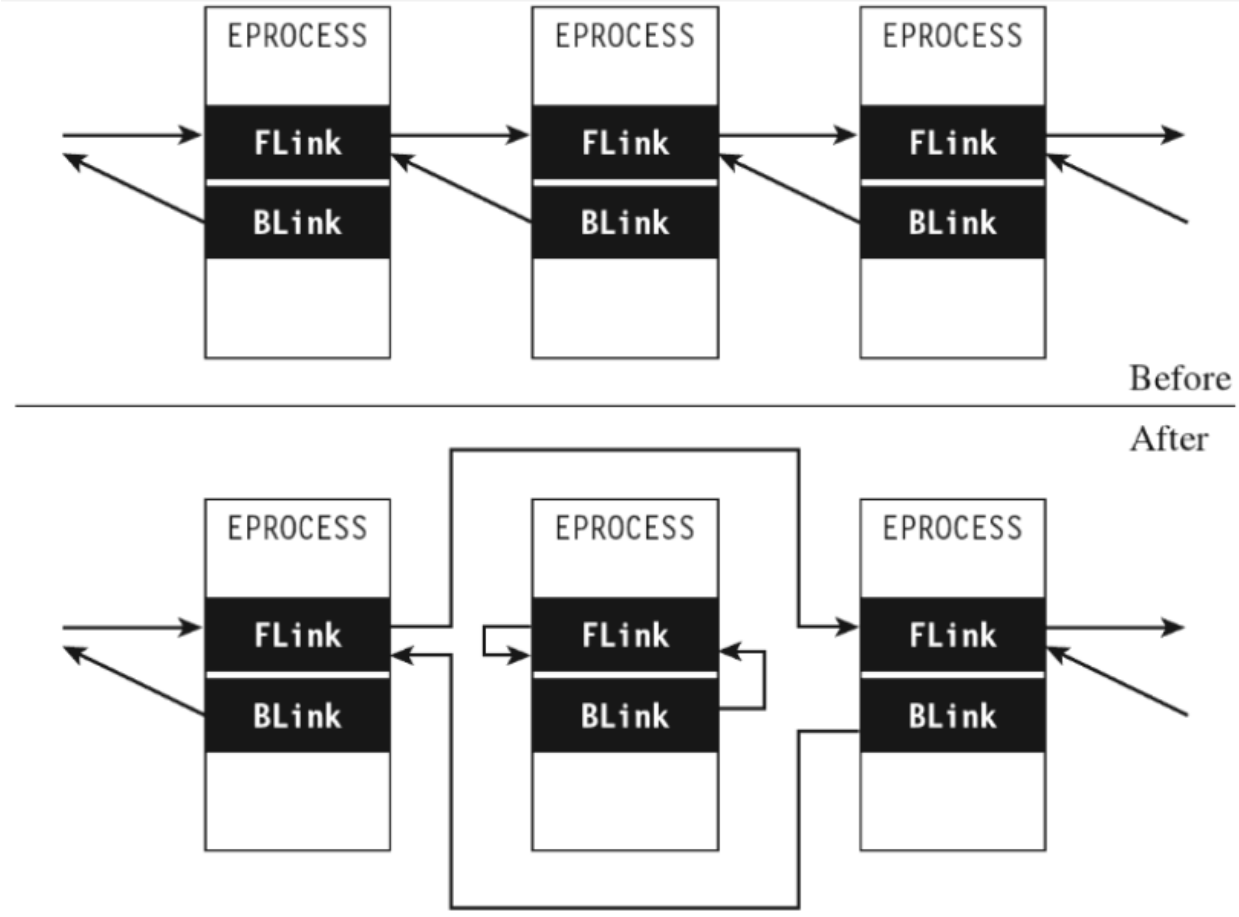
```
INT retrievePid( PCHAR currEP )
{
	INT pid;
	pid = *((INT*)(currEP + EP_OFFSET));
	return pid;
}
```

Once the process id is retrieved we can compare it with the one that we want to hide, and if it does not match we can keep traversing the linked list, in the same way, untill our desired EPROCESS is found or untill the traversal threshold is reached.

Once the relevant EPROCESS is found, now comes the matter of removing it from the list. Firstly we need to gain exlusive access to the kernel structure, so that no other process is modifying it while we remove the desired node from the list. In order to do that we create a Defered Procedure Call (DPC), and add it to the queue, one for each running core of the processor. What this does is that this makes all the cores raise their

IRQL (Interrup Request Levels) to DISPATCH_LEVEL. At this level, the cores cannot process further interrupts unless they have been generated from hardware devices. Since the interrupt to access the EPROCESS linked list will not be generated from a hardware device, we can safely modify the linked list, on our current core with exclusive access.

Now, we will make the pointers of the next and previous nodes point to each other, and make both the Flink and Blink pointers of the current EPROCESS node to point to itself. Such that the following situation will arise:



Now that we have succesfully removed the desired EPROCESS from the linked list, its time to signal the DPC procedures running on all the cores of the processor. This will make the DPC lower their IRQLs to their original values, allowing them to services interrupts normally again, and thus giving our exclusive access to the kernel structure back. Now our desired process will be completely hidden from any task managers, or process managers. This is essential a ring 0, or kernel mode rootkit technique that we have employed here.

Tools/Technologies used:

Windows Driver Kit (C/C++), Microsoft Visual Studio, WinDbg, VMWare Workstation

5.2 Registry Activity Monitor

A driver has been written to intercept and monitor all the registry activity that a process performs in the system. For this purpose we have made use of the CmRegisterCallbackEx function:

The **CmRegisterCallbackEx** routine registers a ***RegistryCallback*** routine.

Syntax

C++

```
NTSTATUS CmRegisterCallbackEx(
  _In_        PEX_CALLBACK_FUNCTION Function,
  _In_        PCUNICODE_STRING Altitude,
  _In_        PVOID Driver,
  _In_opt_    PVOID Context,
  _Out_       PLARGE_INTEGER Cookie,
  _Reserved_  PVOID Reserved
);
```

Parameters

Function [in]
A pointer to the ***RegistryCallback*** routine to register.

Altitude [in]
A pointer to a **UNICODE_STRING** structure. This structure must contain a string that represents the ***altitude*** of the calling **minifilter driver**. For more information, see Remarks.

Driver [in]
A pointer to the **DRIVER_OBJECT** structure that represents the driver.

Context [in, optional]
A driver-defined value that the configuration manager will pass as the *CallbackContext* parameter to the *RegistryCallback* routine.

Cookie [out]
A pointer to a LARGE_INTEGER variable that receives the value that identifies the callback routine. When you unregister the callback routine, pass this value as the *Cookie* parameter to **CmUnRegisterCallback**.

Reserved
This parameter is reserved for future use.

Using this routine, we register a callback that gets called whenever a registry operation is about to take place, is taking place or has taken place.

A filter driver's *RegistryCallback* routine can monitor, block, or modify a registry operation.

Syntax

C++

```
EX_CALLBACK_FUNCTION RegistryCallback;

NTSTATUS RegistryCallback(
  _In_      PVOID CallbackContext,
  _In_opt_  PVOID Argument1,
  _In_opt_  PVOID Argument2
)
{ ... }
```

Parameters

CallbackContext [in]
The value that the driver passed as the *Context* parameter to CmRegisterCallback or CmRegisterCallbackEx when it registered this *RegistryCallback* routine.

Argument1 [in, optional]
A REG_NOTIFY_CLASS-typed value that identifies the type of registry operation that is being performed and whether the *RegistryCallback* routine is being called before or after the registry operation is performed.

Argument2 [in, optional]
A pointer to a structure that contains information that is specific to the type of registry operation. The structure type depends on the REG_NOTIFY_CLASS-typed value for *Argument1*, as shown in the following table. For information about which REG_NOTIFY_CLASS-typed values are available for which operating system versions, see REG_NOTIFY_CLASS.

Then we filter the registry actions based on the process ids that we want to monitor and later the exact activities that we want to be monitored. For the registry, the following activities are being monitored:

- Key creation
- Key deletion
- Key renaming
- Key value creation
- Key value deletion

- Key value modification

For this purpose the following actions/notifications have been set to be monitored in the system:

- RegNtPostCreateKeyEx
- RegNtPreDeleteKey
- RegNtPostDeleteValueKey
- RegNtPreRenameKey
- RegNtPreSetValueKey

Tools/Technologies used:

Windows Driver Kit (C/C++), Microsoft Visual Studio, WinDbg, VMWare Workstation

5.3 File System activity monitor

To monitor the file system activity, a minifilter driver has been written. In a minifilter driver pre and post filesystem operation callbacks are registered, which are called whenever the monitored action is to be performed or has been performed respectively. Following operations on the file system are being monitored:

- File creation
- File deletion

- File modification
- File overwriting
- Folder creation
- Folder deletion

A minifilter provides a set of paths, that can be monitored by registering callbacks for certain operations. The following table shows all the paths, that had to be registered with callbacks and then implemented for their corresponding operations.

File System operation	Registered paths/Major functions
File Creation	IRP_MJ_CREATE
File Deletion	IRP_MJ_CLEANUP IRP_MJ_CREATE IRP_MJ_SET_INFORMATION
File Modification	IRP_MJ_WRITE
File Overwriting	IRP_MJ_CREATE
Folder Creation	IRP_MJ_CREATE
Folder Deletion	IRP_MJ_CLEANUP

File Creation, File Overwriting and Folde Creation:

IRP_MJ_CREATE Major Function, is used to detect the creation of files, by registry pre and post operation routines. When an object is created by a process on the file system, the event is passed to the callback. In the callback first the id of the originating process is checked, if the process is being monitored then further processing is done. The driver

checks if the object being created is a file's unnamed stream or an alternate stream. In the case of an unnamed stream, it means that a file or folder actually has been created. Then it checks whether the object created is a file or a directory and then notifies accordingly. Similarly, it also checks, in the return values of the creation function, during the post operation routine, whether the file created was normal, or was another file overwritten, and once again it notifies accordingly.

File Deletion, Folder Deletion:

There are multiple ways for a file to be deleted. A file can either be marked during its creation (IRP_MJ_CREATE path) with the FILE_DELETE_ON_CLOSE flag. Which when the handle count to the file becomes zero will be deleted, in the IRP_MJ_CLEANUP path. Another way is that via the IRP_MJ_SET_INFORMATION path, the file's FileDispositionInformation class can be set with a boolean flag value of true (boolean DeleteFile). This will also cause the file to be deleted once the handle count to the file becomes zero. All files with a possible deletion pending on cleanup are marked by associating a context stream with them.

Once the handle count to these files become zero, the IRP_MJ_CLEANUP path callback is executed, which detects which of these files have actually been deleted and whether it was a file or a directory and then notifies accordingly.

File Modification:

Whenever a file is written to, i.e. modified, the callbacks of the IRP_MJ_CREATE path are executed. Here based on the process id, it is notified to the user mode driver component, which files were modified by which processes.

Tools/Technologies used:

Windows Driver Kit (C/C++), Microsoft Visual Studio, WinDbg, VMWare Workstation

5.4 Network Activity Monitor

In order to monitor the TCP and UDP endpoints created by the selected processes, Microsoft's IPHelper API has been utilized. The GetExtendedTcpTable() and GetExtendedUdpTable() routines are executed periodically to get the current endpoints. Subsequently, their originating process ids are also acquired and the new endpoints are added to the log reports.

The **GetExtendedTcpTable** function retrieves a table that contains a list of TCP endpoints available to the application.

Syntax

C++

```
DWORD GetExtendedTcpTable(
  _Out_    PVOID pTcpTable,
  _Inout_  PDWORD pdwSize,
  _In_     BOOL bOrder,
  _In_     ULONG ulAf,
  _In_     TCP_TABLE_CLASS TableClass,
  _In_     ULONG Reserved
);
```

The **GetExtendedUdpTable** function retrieves a table that contains a list of UDP endpoints available to the application.

Syntax

C++

```
DWORD GetExtendedUdpTable(
  _Out_    PVOID pUdpTable,
  _Inout_  PDWORD pdwSize,
  _In_     BOOL bOrder,
  _In_     ULONG ulAf,
  _In_     UDP_TABLE_CLASS TableClass,
  _In_     ULONG Reserved
);
```

Tools/Technologies used:

Microsoft Visual Studio, IPHelper API, Win32 API (C/C++)

5.5 Keystroke logger

In order to log the keystrokes, Windows Hooking API has been utilized. The SetWindowsHookEx() is used to set a low level system wide keyboard hook (WH_KEYBOARD_LL).

Installs an application-defined hook procedure into a hook chain. You would install a hook procedure to monitor the system for certain types of events. These events are associated either with a specific thread or with all threads in the same desktop as the calling thread.

Syntax

C++

```
HHOOK WINAPI SetWindowsHookEx(
  _In_  int idHook,
  _In_  HOOKPROC lpfn,
  _In_  HINSTANCE hMod,
  _In_  DWORD dwThreadId
);
```

For each keystroke event, the window handle for the window in foreground is retrieved using GetForegroundWindow() routine. The process id of the foreground application which is receiving the keystroke is then retrieved using GetWindowThreadProcessId()routine. If that retrieved process id is being monitored, the keystroke is logged in the appropriate file, else it is discarded. Further the mapping of the keystrokes to virtual keys, and the detection of small or capital letters and special characters takes place, using other various Win32 API functions and a few macros.

Tools/Technologies used:

Microsoft Visual Studio, Win32 API, Windows Hooking API (C/C++)

5.6 Remote Connection Module (Activity Logger Side)

The communication between the activity logger and the remote client, and all the transfers on the activity logger side, have been implemented using C Sockets. In order to connect with remote client activity logger continuously check for remote client and attempt to connect after every 5 seconds until it succeeds .On the other hand client connects with activity logger and the list of all the connections is maintained. Remote client connects with one of them at a time and communicate.

Tools/Technologies used:

Microsoft Visual Studio, Win32 API, C Sockets (C/C++)

5.7 Remote Configuration Module

.Net's built in libraries were used to modify the retrieved configuration settings of the remote system. In order to get processes running on remote system Win32 API functions OpenProcess() , QueryFullProcessImageName() were used on remote side and the information of this process is sent using window Sockets .Remote client maintains the list of these processes and using this information user can configure the activity logger.

Tools/Technologies used:

Microsoft Visual Studio (C#), Win32 API, C Sockets (C/C++)

5.8 Log Parser

HTML templates were created beforehand on which to map the parsed log data. The logs are parsed using the build in .Net libraries. On the bases of Process Being monitored log parser get activities of each process from log data against the process name. Once the parsing is done, this data is mapped on to the templates using the agility pack and javascript.

Tools/Technologies used:

Microsoft Visual Studio (C#), Agility Pack (C#), HTML, Javascript

5.9 Remote Connection Module (Client Side)

.Net Sockets were used to communicate with the activity logger, get status information and retrieve raw reports from the remote system which were then parsed.

Tools/Technologies used:

Microsoft Visual Studio, .Net Sockets (C#)

5.10 Process Name Retrieval

Most of the logic of the program, requires for it to retrieve the complete name of the process, including its path. This is basically how the process ids have been mapped to the process names. The mapping is of many to one, since one executable can spawn multiple process (by being executed multiple times for example). Thus for example C:\Windows\Notepad.exe can be run multiple times, and the same executable will have multiple running instances with different process ids. Retrieving the process name including the whole image path though is not as simple as it sounds. It requires open a handle to the process, of whose name and full path we want to retrieve and then calling the QueryFullProcessImageName() routine on it. However, a handle to the processes with higher priveleges, such as system processes e.g. csrss.exe, svchost.exe is not possible unless we escalate the priveleges of the opening/calling process itself. This is done by retrieving the token object of the calling process and then adding and enabling the “SeDebugPrivilege” as shown below:

```
If (!LookupPrivilegeValue(NULL, “SeDebugPrivilege”,  &luid))
{
   return FALSE;
}
tokenPrivs.PrivilegeCount = 1;
tokenPrivs.Privileges[0].Luid = luid;
tokenPrivs.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
currentProcess = GetCurrentProcess();
```

```
if (!OpenProcessToken(currentProcess, TOKEN_ADJUST_PRIVILEGES, &processToken))
{
   return FALSE;
}
If (!AdjustTokenPrivileges(processToken, FALSE, &tokenPrivs,
sizeof(TOKEN_PRIVILEGES), (PTOKEN_PRIVILEGES)NULL, (PDWORD) NULL))
{
   return FALSE;
}
```

This allows us to open a handle to all the running process and subsequently retrieve their name and full image path.

Tools/Technologies used:

Microsoft Visual Studio, Win32 API (C/C++)

Chapter 6:

TESTING AND RESULTS ANALYSIS

6. Testing

6.1 Testing introduction:

This software test plan is to provide the description of test cases for Activity Logger with Rootkit describing the scope and approach of intended test activities. This document will describe the test cases for different features of the system.

Software Requirement and Specification document of Activity Logger with Rootkit support this test plan.

6.2 Unit Testing:

Unit testing was performed on each of the modules, during the development and debugging phase, to ensure their correct working as desired, and as required by the specifications.

6.3 Integration Testing:

After the system was integrated, a blackbox testing was performed of the overall system, based on the test cases which will be described below, to ensure that the system is working as intended and as required by the Software Requirement and Specification document.

6.4 Test Items

Following are test items and their version:

Test Item Name	Test Item Version No	Test Type
Overall System	Ver 1	Blackbox
Registry Activity Logging	Ver 1	Blackbox
File System Activity Logging	Ver 1	Blackbox
Network Activity Logging	Ver 1	Blackbox
KeyStroke Logging	Ver 1	Blackbox
Client Listening for online Activity Loggers	Ver 1	Blackbox
Activity Logger remote configuration	Ver 1	Blackbox
Report retrieval and parsing	Ver 1	Blackbox
Generated report review	Ver 1	Blackbox
Stealth	Ver 1	Blackbox

Test Case #1

Name: Overall System

Description: It will test the overall functionality of the system and the working of the system.

Precondition: The activity logger and the remote client are running on their respective systems

Input Values: No input required just the intial exection of the programs

Steps:

1. The Activity Logger client is executed and run on its system
2. The remote client and configurator is executed and run on its system

Expected Output: Both of the programs should neither crash themselves, nor crash the operating environments

Output: Same as the expected outcome

Result: Pass

Test Case #2

Name: Registry activity logging

Description: This will test the registry activity monitoring and logging feature of the program

Precondition: The activity logger and the remote client are running properly on their respective systems

Input Values: Perform some registry activity with a process that is being monitored e.g. Registry Editor

Steps:

1. Connect to the repective activity logger
2. Retrieve the report by pressing the get report button
3. The report will be downloaded and parsed and then opened in the default browser
4. Check whether the performed registry activity is showing in the report at the accurate location

Expected Output: The parsed report must show the performed registry activity at its correct location

Output: Same as the expected outcome

Result: Pass

Test Case #3

Name: File System activity logging

Description: This will test the File System activity monitoring and logging feature of the program

Precondition: The activity logger and the remote client are running properly on their respective systems

Input Values: Perform some file system activity with a process that is being monitored e.g. Windows Explorer

Steps:

1. Connect to the repective activity logger
2. Retrieve the report by pressing the get report button
3. The report will be downloaded and parsed and then opened in the default browser
4. Check whether the performed file system activity is showing in the report at the accurate location

Expected Output: The parsed report must show the performed file system activity at its correct location

Output: Same as the expected outcome

Result: Pass

Test Case #4

Name: Network activity logging

Description: This will test the network activity monitoring and logging feature of the program

Precondition: The activity logger and the remote client are running properly on their respective systems

Input Values: Perform some network activity with a process that is being monitored e.g. Internet Explorer

Steps:

1. Connect to the repective activity logger
2. Retrieve the report by pressing the get report button
3. The report will be downloaded and parsed and then opened in the default browser
4. Check whether the performed network activity is showing in the report at the accurate location

Expected Output: The parsed report must show the performed file system activity at its correct location

Output: Same as the expected outcome

Result: Pass

Test Case #5

Name: KeyStroke logging

Description: This will test the KeyStroke monitoring and logging feature of the program

Precondition: The activity logger and the remote client are running properly on their respective systems.

Input Values: Press any random keys on the keyboard, while a program whose process is being monitored is in focus

Steps:

1. Connect to the repective activity logger
2. Retrieve the report by pressing the get report button
3. The report will be downloaded and parsed and then opened in the default browser
4. Check whether the keystrokes made are showing in the report at the accurate location

Expected Output: The keystrokes sent to the selected process must be shown correctly and their appropriate locations in the final parsed report

Output: Same as the expected outcome

Result: Pass

Test Case #6

Name: Client Listening for online Activity Loggers

Description: This will test whether the client is listening properly, and the activity logger broadcasting attempting to connect properly

Precondition: The activity logger and the remote client are running properly on their respective systems

Input Values: Press the start listening button on the Remote Client

Steps:

1. Press the “Start Listening” button on the remote client
2. Wait for the “Online Activity Loggers list” to be populated

Expected Output: The Activity Loggers currently online should start showing up in the list

Output: Same as the expected outcome

Result: Pass

Test Case #7

Name: Activity Logger remote configuration

Description: This will test whether the configurations of the Activity Logger are being updated remotely correctly

Precondition: The activity logger and the remote client are running properly on their respective systems

Input Values: Select an online Activity Logger from the list and change its configuration i.e. change the list of processes that is being monitored by it

Steps:

1. After changing the settings of the Activity Logger, press the “Save” button
2. Wait for the “Saved” dialog box to appear

Expected Output: The Activity Logger’s configuration should now have been changed and saved according to the modifications made from the remote client and now the new list of processes should start being monitored and show up in the final report.

Output: Same as the expected outcome

Result: Pass

Test Case #8

Name: Report retrieval and parsing

Description: This will verify that the report is being downloaded correctly and then being parsed as required

Precondition: The activity logger and the remote client are running properly on their respective systems

Input Values: After selecting an online Activity Logger, press the “Get Report” button

Steps:

1. After pressing the “Get Report” button the report will start downloading
2. Wait for the report to be downloaded
3. Wait for the report to be parsed

Expected Output: A correctly parsed report should open up in the default browser of the operating system

Output: Same as the expected outcome

Result: Pass

Test Case #9

Name: Generated report review

Description: This will test whether the report that has been generated is correct according to the intended format, and does not have any misformatting or broken links in it

Precondition: The activity logger and the remote client are running properly on their respective systems

Input Values: Retrieve a report from a selected Activity Logger and let it be parsed

Steps:

1. Press the “Get Report” button
2. Wait for the report to be downloaded
3. Wait for the report to be parsed
4. Wait for the report to be opened in the default browser of the operating system

Expected Output: The report should be correctly formatted, showing all the performed activity, without any broken links, or out of alignment elements

Output: Same as the expected outcome

Result: Pass

Test Case #10

Name: Stealth

Description: This will test the rootkit part of the Activity Logger, that whether its process is being hidden properly from the task/process managers etc.

Precondition: The activity logger is running properly on their respective systems

Input Values: No input required just the normal initial execution of the activity logger program

Steps:

1. Run the activity logger
2. Open up any task managers/process managers on the system (system default or third party)

Expected Output: The Activity Logger task should not be showing in the task manager, and its process should not be showing in either the process manager, or the running services section

Output: Same as the expected outcome

Result: Pass

Chapter 7:

CONCLUSION AND FUTURE WORK

7. Conclusion and Future Work

The goal of the project was to create a process activity logger, that can monitor the behaviour of the selected processes in a system remotely. The generated reports will then prove to be helpful in determining what the process was actually doing behind the scenes, what activity it was performing, what modifications it was making to the system etc. Another goal was to keep the process itself hidden in the system.

Further the same techniques utilized in this project could be expanded upon to block certain operations in certain locations in the system. Locations which might contain sensitive data or are protected. For example, we might want to add the feature, that a certain area of the file system can only be accessed with a certain program, and by nothing else, this could be added as a feature. Or a feature, that certain words or patterns of characters cannot be written to the system, or copied or even read etc could also be added (both process based and generic).

Similarly, the DKOM rootkit techniques that have been used can also be applied to hide the network connections of any process in the system and also hide any drivers in the system, or allow for privilege escalation of other less priveleged processes.

8. Appendix A: User Manual

Installing the activity logger:

- Load the rootkit driver ALHider.sys and MiniFilter.sys using the driver loader utility
- Install double click on the ALogger.exe to run the activity logger

Running the remote client:

- Run AClient.exe and press the start listening button, all the online Activity Loggers will show up in the list on the left side.
- Select any Activity Logger Client and all its information will be displayed on the main screen

Configuring an Activity Logger:

- After selecting an Activity Logger, press the “Configure Activity Logger” button and the configuration window will pop up.
- All the running processes on the remote system will be displayed on the left side of the screen. Highlight any process you want to monitor and send it to the list on right side of “currently monitored processes” by click the “>>” button to the right.
- After having selected the desired processes, click the save configuration button.

Retrieving reports:

After selecting an Activity Logger, click on the “Get Report” button. The reports will start downloading. After the download is finished, client will automatically parse the reports and display them in the default browser of the system.