

A Behavior Based Automatic Security Monitor

A Project Report
Presented to
The Faculty of the College of
Engineering
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By
Maulik Thaker
Kunal Vyas
Venkatesh Babu
December 2007

Copyright © 2007
Maulik Thaker
Kunal Vyas
Venkatesh Babu
ALL RIGHTS RESERVED

APPROVED FOR THE COLLEGE OF ENGINEERING

Dr. Frank C Lin, Project Advisor

Dan Harkey, Director, MS Software Engineering

Dr. Sigurd Meldal, Chair, Computer Engineering Department

ABSTRACT

In this project, we attempt to approach the solution to system security by researching into techniques that address system threats by analyzing the behaviors of applications. The major goal of this project will be to come up with behavior based intelligent system that can automate the system security, appropriately and without any significant administrator help. The aim of this project will be preventing false alarms and generate a correct warning based on the behavior intelligence.

The focus will be to monitor the system call sequences of the process, and identify malicious activities and threats that take place on the system. The protection system would monitor the system activity; identify malicious activities and threats through the content analysis. One of the main tasks would also be to minimize false positives by researching on various normal activities in the system and distinguish clearly what is an unacceptable behavior. As the part of this project, behavior of some of the popular applications like Internet Explorer, Mozilla Firefox and Outlook Express are captured by creating the learning database by running application in well controlled normal environment. This learning database is used to study the application's behavior based security monitor. Any suspected abnormal behavior of the application based on the sequence of system calls is reported as fault message.

Acknowledgments

The authors are deeply indebted to Professor Dan Harkey, Professor Frank C Lin and Mr. Swar Shah for their invaluable comments and assistance in the preparation of this study.

Table of Contents

ABSTRACT	iv
Chapter 2. Project Overview	1
2.1 Introduction	1
2.2 Proposed Areas of Study and Academic Contribution.....	2
2.3 Current State of the Art	3
Chapter 3. Project Architecture	7
3.1 Introduction	7
3.2 Architecture Description	8
3.3 System Interface Diagram	9
3.4 System Interface Description	10
3.5 Architecture Subsystems	10
3.6 Detours Module	11
3.7 Content Analyzer Module	11
3.8 Database - logs	11
3.9 The Notification Module	12
Chapter 4. Technology Descriptions	13
4.1 Client Side Technologies.....	13
4.1 Server Side Technologies	14
4.3 Data-Tier Technologies	15
Chapter 5. Project Design	16
5.1 Client Design	16
5.2 Middle-Tier Design	18
5.3 Use Cases.....	19
5.4 Sequence Diagrams	22
5.5 Use Case 1 : Sequence Diagram Description	23
5.6 Use Case 2 : Sequence Diagram Description	24
5.7 Use Case 3 : Sequence Diagram Description	25
5.8 Activity Diagrams	25
5.9 Class Diagram	28
5.10 Description:	29
5.11 Data-Tier Design	30
5.11.1 Application behavior patterns	30
5.11.2 Event Notification Log.....	30

Chapter 6. Project Implementation.....	31
6.1 Serializable variant API parameter passing.....	31
6.1.1 APICallAnalyzer (Server)	42
6.2 GUI Implementation details	45
6.3 The Analyzer Module.....	46
6.3.1 Training Mode.....	46
6.3.2 Analysis Mode.....	47
6.4 Program Code Details.....	50
6.5 Test Cases Implementation.....	55
Chapter 7. Performance and Benchmarks	63
Chapter 8. Deployment, Operations, Maintenance	76
8.1 Deployment	76
8.2 Operations.....	78
Chapter 9. Summary, Conclusions, and Recommendations.....	82
9.1 Summary.....	82
9.2 Conclusions	83
9.3 Recommendations for Further Research	85
Glossary	87
References.....	88

List of Figures

Figure 1: Overall System Architecture	7
Figure 2 : System Interface Digaram	9
Figure 3 : System Modules based on the program flow	10
Figure 4: User Interface Design for BASM.....	16
Figure 5 : Start Application dialog for BASM.....	17
Figure 6 : Sequence Diagram-I.....	22
Figure 7 : Sequence Diagram-II.....	23
Figure 8 : Sequence Diagram-III	24
Figure 9 : Activity Diagram for Notification.....	26
Figure 10 : Activity Diagram for Defining Normal Application Behavior	27
Figure 11 : System Class Diagram.....	28
Figure 12 : Application, APICallAnalyzer Server with detour.....	32
Figure 13: Application API Call without Detour.	33
Figure 14: Application API Call with Detour	33
Figure 15: Communication - GUI Remote Client/Server (APICallAnalyzer).....	42
Figure 16: GUI component.....	45
Figure 17: Chart – IE Deviation (window 4-25).....	64
Figure 18: Chart – IE Learning DB (window 4-25).....	65
Figure 19: Chart – IE Window size vs Normal Database	65
Figure 20: Chart – IE Percentage Deviation vs Window Size	66
Figure 21: Chart – Fire Fox Deviation (window 4-25).....	67
Figure 22: Chart – Fire Fox Learning DB (window 4-25)	68
Figure 23: Chart – FireFox Window size vs Normal Database.....	68
Figure 24: Chart – Fire Fox Percentage Deviation vs Window Size	69
Figure 25: Chart – Outlook Express Deviation (window 4-25)	71
Figure 26: Chart – Outlook Express Learning DB (window 4-25).....	71
Figure 27: Chart – Outlook Express Window size vs Normal Database	72
Figure 28: Chart – Outlook Express Percentage Deviation vs Window Size.....	72
Figure 29: Event Log (Each Run).....	73
Figure 30: Chart – Time per run vs Window Size	74
Figure 31: Chart – Deviation window (4-7).....	75
Figure 32: BASM.: Executables Source Code.....	77
Figure 33: BASM : GUI connect/disconnect to server.....	78
Figure 34: BASM Connection status	79
Figure 35: BASM : Application	79
Figure 36: BASM : Log Utils.....	80
Figure 37: BASM : API log files.....	80
Figure 38: BASM : Adding Percentage Deviation to Normal Behavior	81

List of Tables

<i>Table 1: Use Case-1</i>	19
<i>Table 2: Use Case-2</i>	20
<i>Table 3: Use Case-3</i>	21
<i>Table 4: Schema for Patterns Db</i>	30
<i>Table 5: Schema for Event log file</i>	30
<i>Table 6: Test Case- Detoured Outlook Express</i>	56
<i>Table 7: Test Case- Detoured Web browsers</i>	57
<i>Table 8: Test Case- Content Analyzer Algorithm Check</i>	58
<i>Table 9: Test Case- Normal Db Creation</i>	59
<i>Table 10: Test Case- Validate Notification</i>	60
<i>Table 11: Test Case- Validate Auto Entry in Event Log</i>	61
<i>Table 12: Test Case- Validate Control Enabling</i>	62

Chapter 2. Project Overview

2.1 Introduction

The purpose of this document is to summarize the research, analysis, design, and the implementation-plan for the system - “Behavior Based Automatic Security Monitor” implementation project.

Attempts have been made in the past to manage the system security by logging various system calls and events and displaying it to the administrator or the system user; approach taken in [3]. This kind of system monitors is not really useful unless you manually scrutinize the log and detect the intrusions. This project tries to overcome these limitations using the detour-library for system calls interception which gives more functionality to the users have some development background.

The main goal of this project is -

“To research, design and implement an intelligent system that monitors and logs the critical behavior of application, analyze the content and reports the adjudged malicious activity.”

The tool will monitor the system activities, file-system manipulation and the system registry to achieve its goal. This document contains various features of the project and the research work performed to fulfill the requirements. The document discusses the overall system architecture in general and discusses the system side module of the tool at length. It is followed by the implementation strategy for the project.

2.2 Proposed Areas of Study and Academic Contribution

According to Iowa State's Jacobson, a typical behavior-based system includes a computer that can capture data packets via a network card and feed them into a stream assembly element, which puts the packets into correct order; a behavior engine uses a set of algorithms to examine the data stream. If the engine detects a probable problem, it initiates corrective measures and notifies the alert element, which informs a system administrator. A reporting element then obtains incident information stored in a database to issue a report for further examination.

There has been a master's project carried out by a student at San Jose State University. It was fascinating towards our research project. The documentation just ended where our project begins. Concisely talking, we have to explore more towards the behavior securing the network unlike the static behavior of the incoming and outgoing system calls specifying the nature of the attack. The more precise task for the project would be to intelligently sort out the false calls for an alert from the system calls. The final aim of the project would be to protect the sensitive data from the network and its assets. These may also include some of the unauthorized utilization of resources, DOS attacks and diverted information (eg. Sniffing and spoofing).

A research paper on 'System Call Monitoring Using Authenticated System Calls' written by Mohan Rajagopalan, Matti Hiltunen, Trevor and Richard in IEEE, shows how an authenticated system call uses additional authentication code which assures the reliability of

the policy of the system call arguments. Kernel uses this information to check for the valid system calls. The version of the application in which regular system calls have been replaced by authenticated calls is generated automatically by an installer program that reads the application binary, uses static analysis to generate policies, and then rewrites the binary with the authenticated calls. This paper presents the approach, describes a prototype implementation based on Linux and the Plto binary rewriting system, and gives experimental results suggesting that the approach is effective in protecting against compromised applications at modest cost

Another thesis report by Debin Gao has worked on different approaches to eradicate abnormal system calls and tried to identify the normal behavior and detecting the malicious activities in the system calls. The report was ‘Gray-Box Anomaly Detection using System Call Monitoring’. In his thesis he has explored two novel approaches for constructing the normal behavior model for anomaly detection.

2.3 Current State of the Art

The idea behind the behavior based automatic security would be to automatically manage the security characteristics of the system and try to reduce human intervention as far as possible. The current systems working on the behavior based network security and system monitoring works on the basis that it learns the normal behavior of the traffic and the systems. This approach recognizes attacks based on what they do, rather than whether their code matches strings used in a specific past incident. In the past, behavior-based security has

been too expensive and too unfamiliar to most IT workers to be widely adopted. Several vendors are thus beginning to make behavior-based security widely available to organizations via services, appliances, and software products. And some ISPs are protecting their entire networks via behavior-based services. However, widespread adoption of behavior-based security faces numerous obstacles, including complexity and a higher number of false positives than signature-based systems.

Currently, there are security tools available that scan for the vulnerability of the system and network. Nessus, is a Unix vulnerability evaluation tool. These tools have the key features that perform security checks on local and remote clients/servers. Some more tools include Wireshark, Snort (open source IDS), and Netcat. These tools do not perform security as and when it happens based on the abnormal behavior of that application either on the system or the network. Hence, there have been developments with the tools and certain plug-ins are also available that make it possible. Our concept is different in a way that not only developing some intelligent system that captures the network and system abnormal behavior, but also analysis captured data with learning database(created using previous normal runs) and reports when abnormal behavior is found. Operator has the option to add the previous abnormal behavior information into learning database so that next time same behavior of the application is treated as normal.

The drawback of behavior-based network security software is that they can generate a much higher level of false alerts. The advantage is it can spot and block new threats, based on their behaviors, before a menace is identified and assigned a threat signature and name.

The paper presented by Dr. Frank Lin et al. have proposed a solution for preventing outgoing spam. The approach was totally different from the Microsoft's spam prevention methodology. Microsoft uses HIP (Host Identity Protocol) challenges to block outgoing spam, the other would be based on computational challenges and the last one would be based on the paid subscriptions. Comparing these, The document presented by Frank Lin would be an extension of Host-based Intrusion Prevention System. Here the Message Intent ability Scoring Algorithm was proposed for preventing outgoing spams. The Behavior based Network Monitoring would detect all the outgoing activities based on the port scan and system calls.

Stephanie Forrest, Professor and Chairman of Computer Science at University of New Mexico has developed a software using sequences of system calls for intrusion detection using different algorithms to generate patterns of system calls. Along with detection, Forrest also utilized the application behavior for modeling intelligent immune system that immunizes itself against various attacks. Apart from her own work, Forrest has laid foundation for several researchers that plan to work on application behavior and intrusion detection in the field of computer security.

Other approach taken is defining policies of system calls for security level - one discussed by Niels Provos for improving host security wherein he discusses Sysrtrace facility, its design and analysis. This deals with restricting certain system calls to run thereby increasing security level.

Apart from generating patterns and observing system call sequences, a further step taken is performing the argument and the data analysis in the system calls. Stefano Zanero, CTO and founder of Secure Network has explored this approach for anomaly detection. Although the approach is not suitable for run time analysis, never the less – it proves to be useful for offline analysis.

Apple has a commercial tool – Shark 4 which has the system call tracing facility but it does not provide with the automated analysis of this trace and leaves to the user to work.

Chapter 3. Project Architecture

3.1 Introduction

The system monitor can be divided into two major phases- learning and analysis. Both the phases take user behavior and activity into account for the threat detection instead of relying on the traditional signature based approach. The system requires major interaction with the Windows kernel and would utilize and modify the available Win32 API as needed.

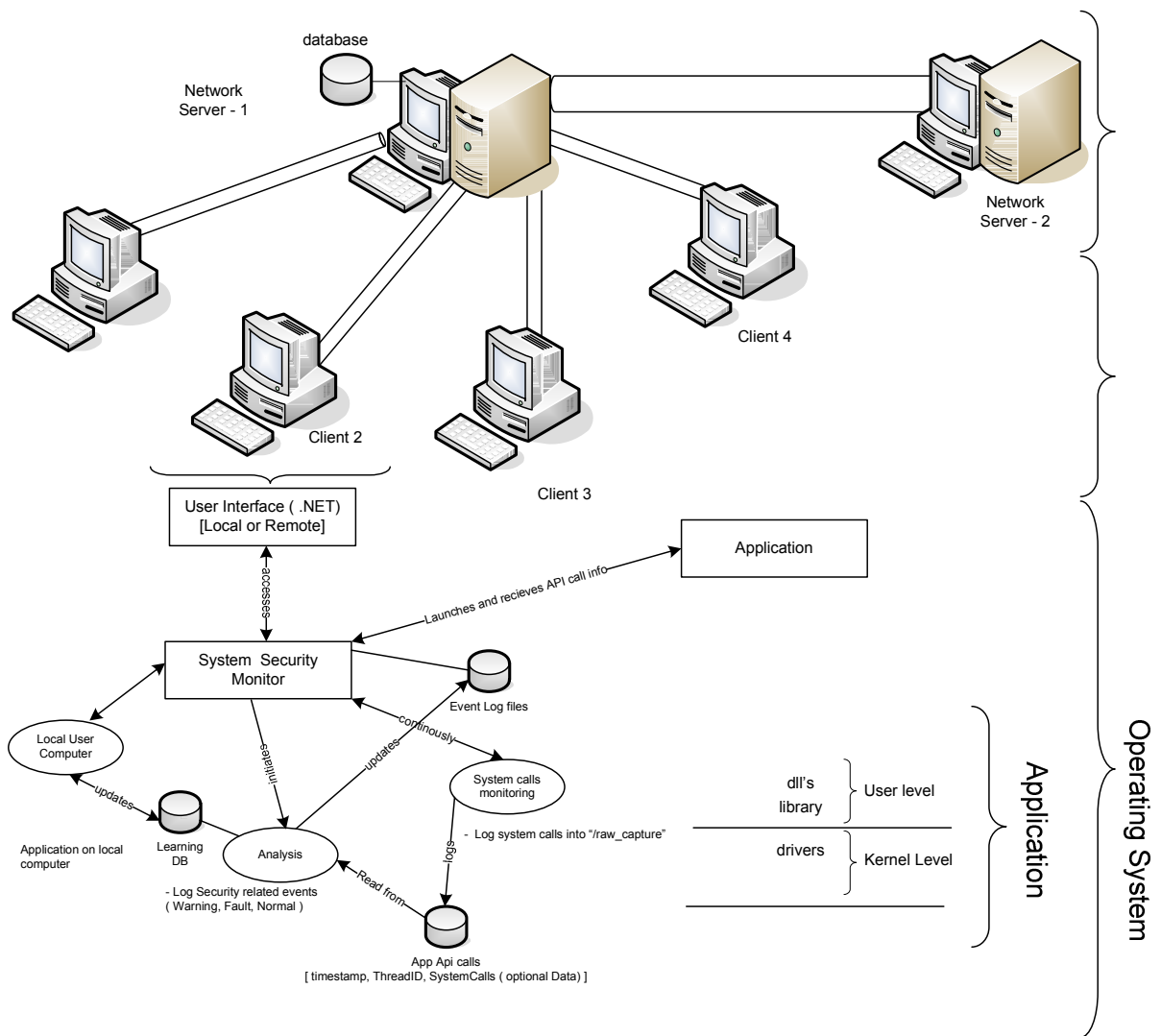


Figure 1: Overall System Architecture

3.2 Architecture Description

The system monitor application runs on Workstation and servers and performs system call monitoring, event monitoring and notification. On a large scale, administrative servers can connect the global users and secure the whole system and the network as a whole.

The main purpose of the system module is to capture inconsistent behavior of the application, DLL's, libraries and different modules on the standalone system. There is a hierarchy of the clients which are connected to a central server. The clients and server are interconnected to form the network. Any machine or host updates the log file or the database file which can then be shared in the network. This helps in developing behavior based intrusion monitoring.

The application is though installed in every host and server machine but the database is shared among the machines. The Kernel consists of the drivers required for the application to run upon and is needed for the privileged services.

At the user level, there are DLL's and libraries which can be used from the application provided by Detours. The application is independent from the OS, because it can be used to run on the remote machine which may be Windows mainframe.

3.3 System Interface Diagram

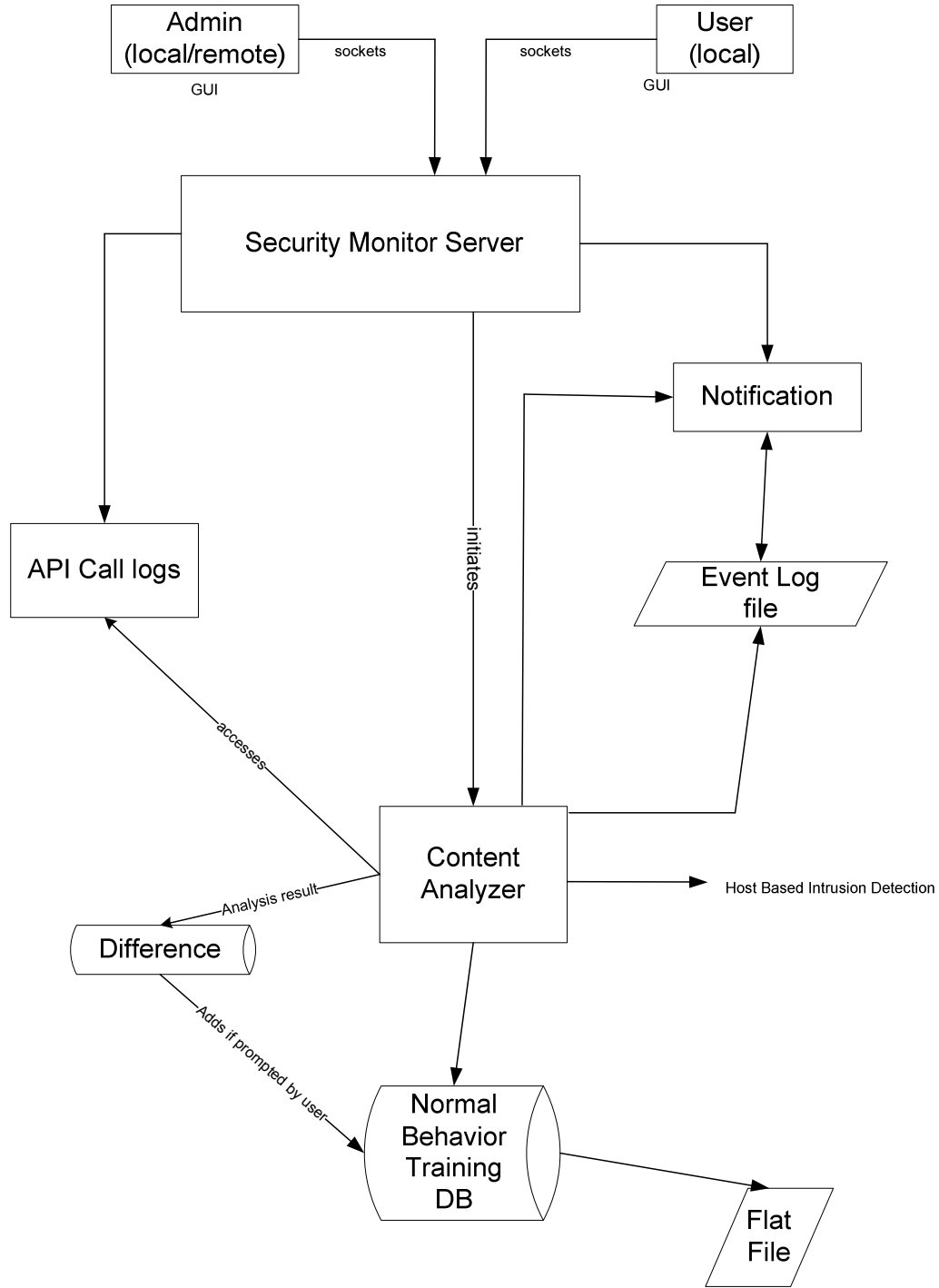


Figure 2 : System Interface Diagram

3.4 System Interface Description

- The system consists of security monitor server which controls and communicates with content analyzer, and event logger.
- Security Monitor Server accepts multiple client connections for browsing event log and configuring Network Security Monitor.
- Content Analyzer module creates learning database for application based when application is running in training mode.
- Content Analyzer module analyzes the application log file when application is running in analysis mode.
- Event Logger module stores events into database (flat file), and also retrieves events on request.

3.5 Architecture Subsystems

The functioning of modules based on the program flow can be pictorially depicted as below-

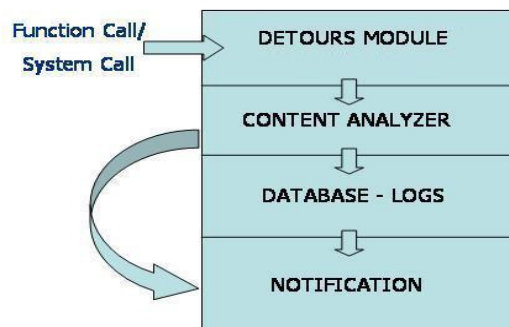


Figure 3 : System Modules based on the program flow

3.6 Detours Module

This module would be the customized API's of the Detours API based on the requirement. The Detours library contains codes that would intercept the normal WIN32 function call(system call) and re-write the in-memory code for the function and give the control to user defined functions. Later on when the processing is finished, it would pass back the control and execute the routine function call. The detours mechanism uses the trampoline design to preserve the un-instrumented target function, which would store couple of next instructions before passing the control to the user define function.

3.7 Content Analyzer Module

The Content Analyzer module is the brain of our system monitor. It would contain set of rules formed from behavior analysis of the application and these rules will be applied on the information passed by the upper module in order to detect anomalous activity. The plan is to feed it with specific set of rules and conditions which the module will utilize to perform the static as well as dynamic behavior analysis.

3.8 Database - logs

The lower module – database logs - will be storing some non-trivial information that can be useful for static analysis provided from the upper modules. Also content analyzer will depend on this particular database for an application to define its normal or abnormal behavior. In short – each application should have its own database defining normal behavior of itself and the logs will be general and shared by the system applications.

3.9 The Notification Module

The notification module stores results of the process that our system monitor carries out.

This module performs the representation of the activity and information to the user is sent as a threat, alert, normal event or miscellaneous based on classification.

Chapter 4. Technology Descriptions

The system monitor is a client server application with client sending the messages of open different applications to the server and sever traces this applications behavior. The software is developed using Microsoft's .Net technology using Visual Studio 2005 and Open source Perl technology. Microsoft Visual Studio provides with advance set of integrated development environment that has features that power developers to write, maintain and debug the code efficiently. The data is stored in structured flat files which prevents expensive overheads of database connection and transactions.

4.1 Client Side Technologies

The client development is carried using Windows Forms 2.0 of the Microsoft .Net framework and Microsoft Visual Studio 2005. Windows forms provide developers with the rich set of controls that can used to develop a clean graphical user interface. With each control – there are specific set of events that are triggered based on the user interaction with the control. Based on these events – server can process requested information and pass it to the client. With our system, we required a user interface that just takes few inputs from user and based on this take appropriate actions at the server end. Also the log information and the notification messages send to client has to be displayed. Windows forms provided all the controls required for our software with very clean interfaces that could be implemented based on the requirements. Apart from being easy to use and developer productive, Windows forms technology comes as a part of .Net framework in which our major

development was done. It wouldn't be wise to add another dependency just for GUI development and not use this feature.

4.1 Server Side Technologies

At the server side, there are two major technologies – Visual C++ .Net and .Net Remoting technologies available in Microsoft Visual Studio 2005 development framework and Perl which is an open source technology.

Visual C++ provides with rich set of libraries like C- Run time (CRT), Standard C++ library and many development interfaces for windows based applications. C++ is most widely used systems-level language and Microsoft's Visual C++ facilitates this process with its advance tools that can be helpful in developing complex systems applications.

Remoting technology enables the distributed application development by providing rich set of application programming interfaces for communication among different components of your system. These components, whether residing on the same system or spread across the network, talk using remote objects which is independent of specific client or server application module and the communication mechanism. Any application can host these remoting objects and can serve any client by configuring itself and the application with the remoting capabilities.

Perl is an open source technology widely used for report processing and data manipulation for better sense. It comes with variety of features from other programming languages. Perl has libraries that come with rich set of API's with file and text processing functions, providing minimal effort development. It is formed with several advanced features borrowed from C, LISP, AWK and has automatic memory management reliving programmer from memory leaks checks.

4.3 Data-Tier Technologies

We used comma separated values (csv) file format as our data tier. CSV data presentation can be easily deployed on all the platforms and provides a simple tabular representation of the data. Data is represented in a tables – with fields and columns separated by comma while the records and rows separated by newlines in the data. The main advantages of using CSV data representation is less overhead and ease of deployment.

Chapter 5. Project Design

5.1 Client Design

BASM Client aims to provide users with simple interface that can monitor application behavior and analyze it. The plan is to implement the GUI using windows forms technology and utilize the rich set of controls provided. The design simple and easy to control, and at the same time providing with the necessary information that is required. Following are the snapshots of user interface design.

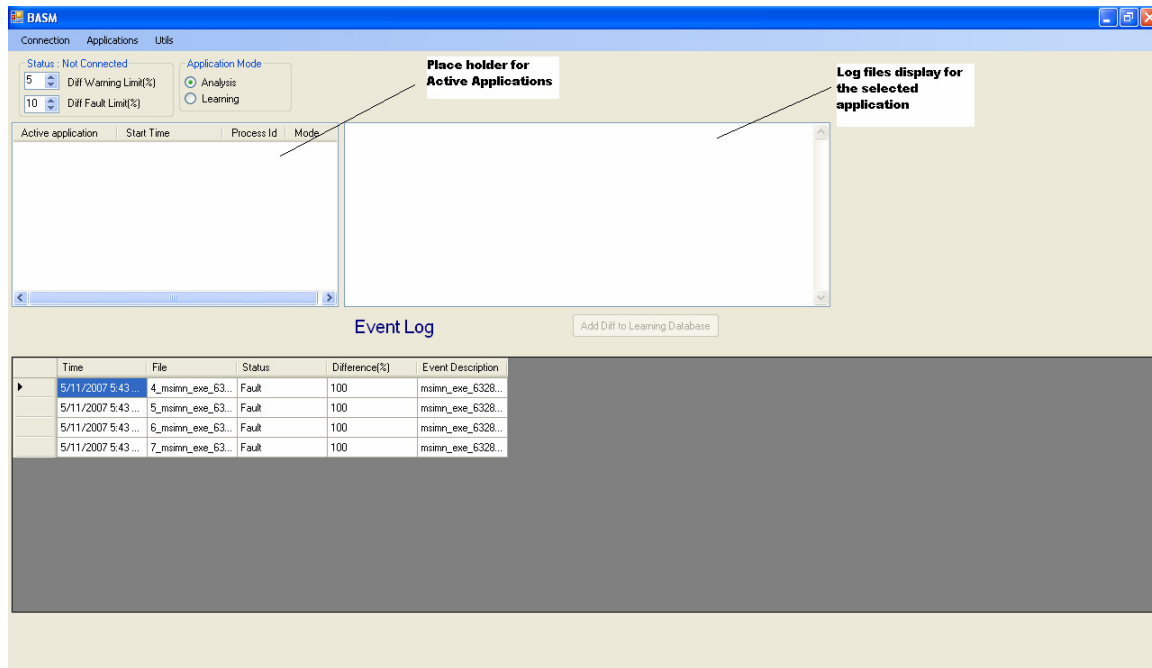


Figure 4: User Interface Design for BASM

The three place holders give out different information of the applications run by the user currently. The first on the left-top shows the list of all the active applications. Next to it would display the log file of the application in the run time.

At the bottom shows all the events in the main Event log.

The other interfaces are to open the application – which is the standard file dialog as shown below

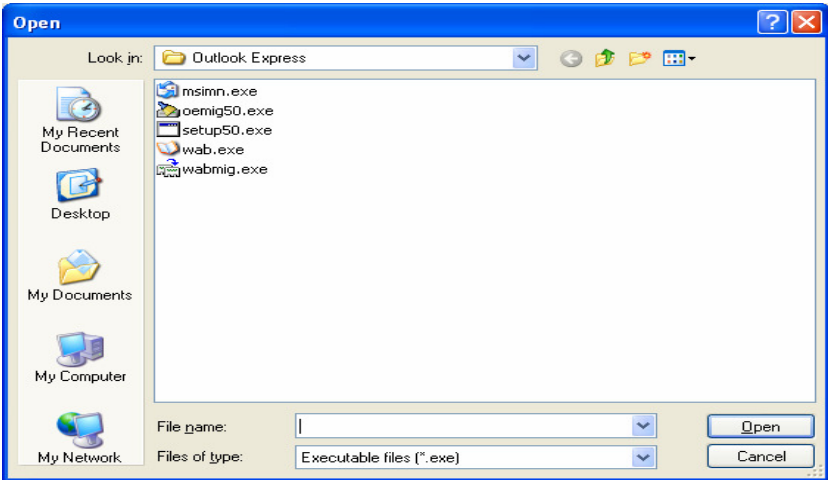


Figure 5 : Start Application dialog for BASM

The user interface provides with updated information of the operations and gives rich set of controls to carry out operations.

5.2 Middle-Tier Design

Middle tier includes the application client that has remoting capabilities that interacts with the client user interface to send the commands to the server. Server basically can be divided into two major parts – the core server that is listening to the clients requests and serving it, while the other is the Content Analyzer which is only responsible for analyzing the raw data sent to it by the server.

To simplify the application and gain its basic understanding from a high level – we begin with giving 3 major use cases of the application system along with their sequence diagrams based on the major components interacting. After that we have another dynamic model that would simulate behavior of system during two major phases – Notification and Defining application behavior. We make use of activity diagrams to accomplish this task.

Following the dynamic modeling we give static model – Class diagram for the application. Since the development involved object oriented C++ along with run time C, the class diagram becomes handy in developing major modules of the system. Following are the models for the system that enabled us to implement the system in a clean and easy manner.

5.3 Use Cases

Use Case Number	UC#1	
Use Case Title	To notify the application behavior	
Actors and Roles	Actor	Role
	User	User
Preconditions	The user must be connected to the server.	
Use Case Description (Steps)	<u>Steps:</u> <ol style="list-style-type: none"> 1. Set the mode – ‘Analysis’ mode on the GUI. 2. Click the ‘Start Application’ to open the application selection dialog. 3. Select the application for analysis. 4. Perform the desired operation. 5. Close the application. 6. Check the Event Log at bottom for the notification 	
Alternative Solution (Error Case)	<ol style="list-style-type: none"> 1. The system will begin analysis once the application is closed. 2. One can select the log file and perform manual analysis without running application also. 	
Post Conditions	The system will have the notification entry as – ‘Normal’ – ‘Fault’ or ‘Warning’ based on the analysis.	

Table 1: Use Case-1

Use Case Number	UC#2	
Use Case Title	To define normal behavior of a selected application	
Actors and Roles	Actor	Role
	User	User
Preconditions	The user must be connected to the server.	
Use Case Description (Steps)	<u>Steps:</u> <ol style="list-style-type: none"> 1. Set the mode – ‘Learning’ mode on the GUI. 2. Click the ‘Start Application’ to open the application selection dialog. 3. Select the application of interest. 4. Perform the desired operations. 5. Close the application. 6. Check the event log for notification. 7. Go to folder - ~\SecurityMonitor\Data\Application_Name 8. Check the .csv file for the application containing patterns of system calls. 	
Alternative Solution (Error Case)	None.	
Post Conditions	<ol style="list-style-type: none"> 1. The system will have the notification entry with ‘Learning Mode’ type. 2. A file with application name will be created that contains application’s normal behavior definitions. 	

Table 2: Use Case-2

Use Case Number	UC#3	
Actors and Roles	Actor	Role
	User	User
Preconditions	<ol style="list-style-type: none"> 1. The user must be connected to the server 2. The system should have generated 'Fault' or 'Warning' for the application behavior 	
Use Case Description (Steps)	<u>Steps:</u> <ol style="list-style-type: none"> 1. Check the Event log for the 'Fault' or 'Warning' with corresponding notification time. 2. Select the log on the GUI. 3. Click the enabled – 'Add Diff to Learning Data' button 4. Check for the event notification in the event log. 	
Alternative Solution (Error Case)	If the behavior was already added previously then it will not be added to the database.	
Post Condition	The abnormal behavior which was legal for user is added to the applications normal behavior database.	

Table 3: Use Case-3

5.4 Sequence Diagrams

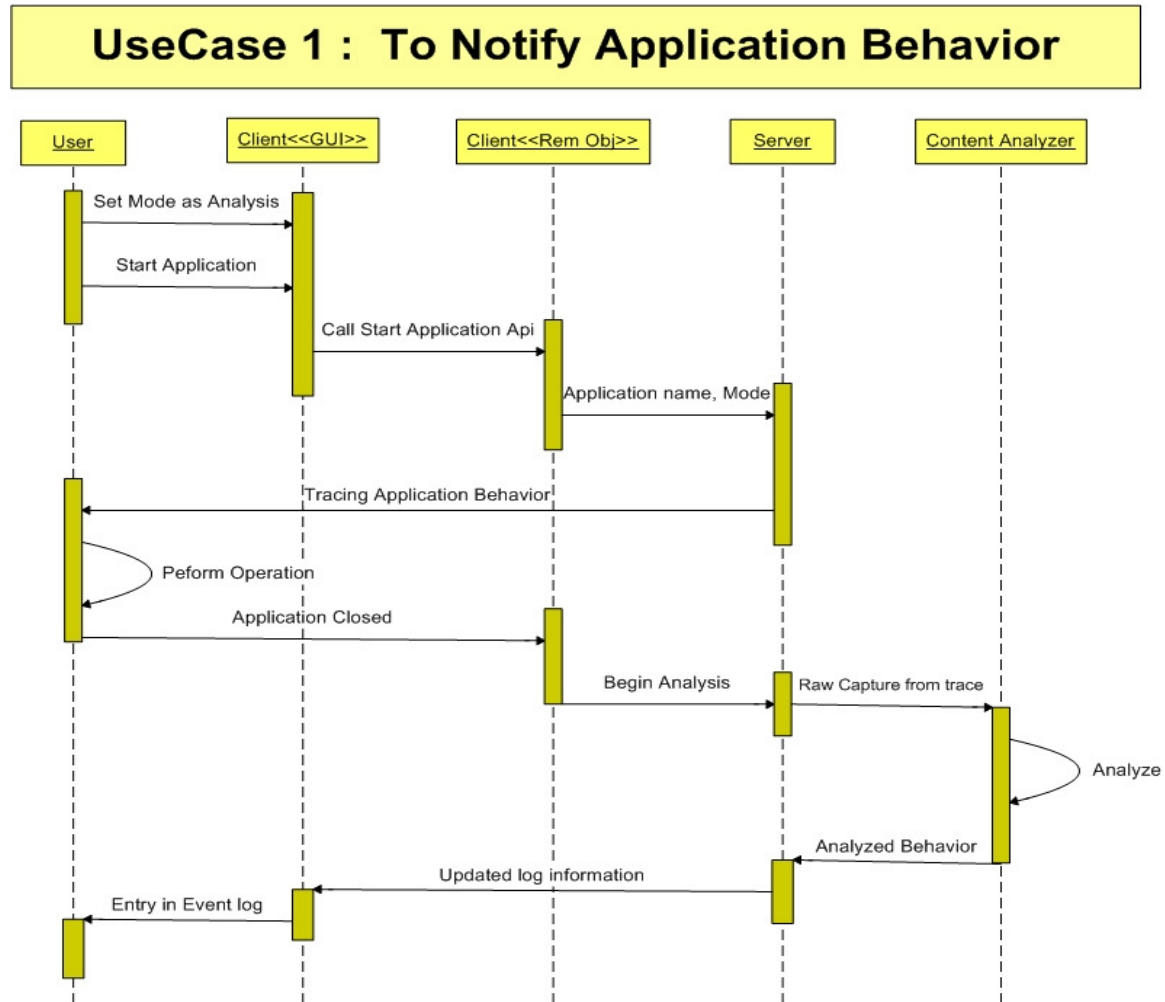


Figure 6 : Sequence Diagram-I

5.5 Use Case 1 : Sequence Diagram Description

The major components taking part during this use case are User from utilization side and Client (GUI), remoting object of client, Server (listening requests) and Content Analyzer from the system side. The message passing among the components is visible from the above diagram.

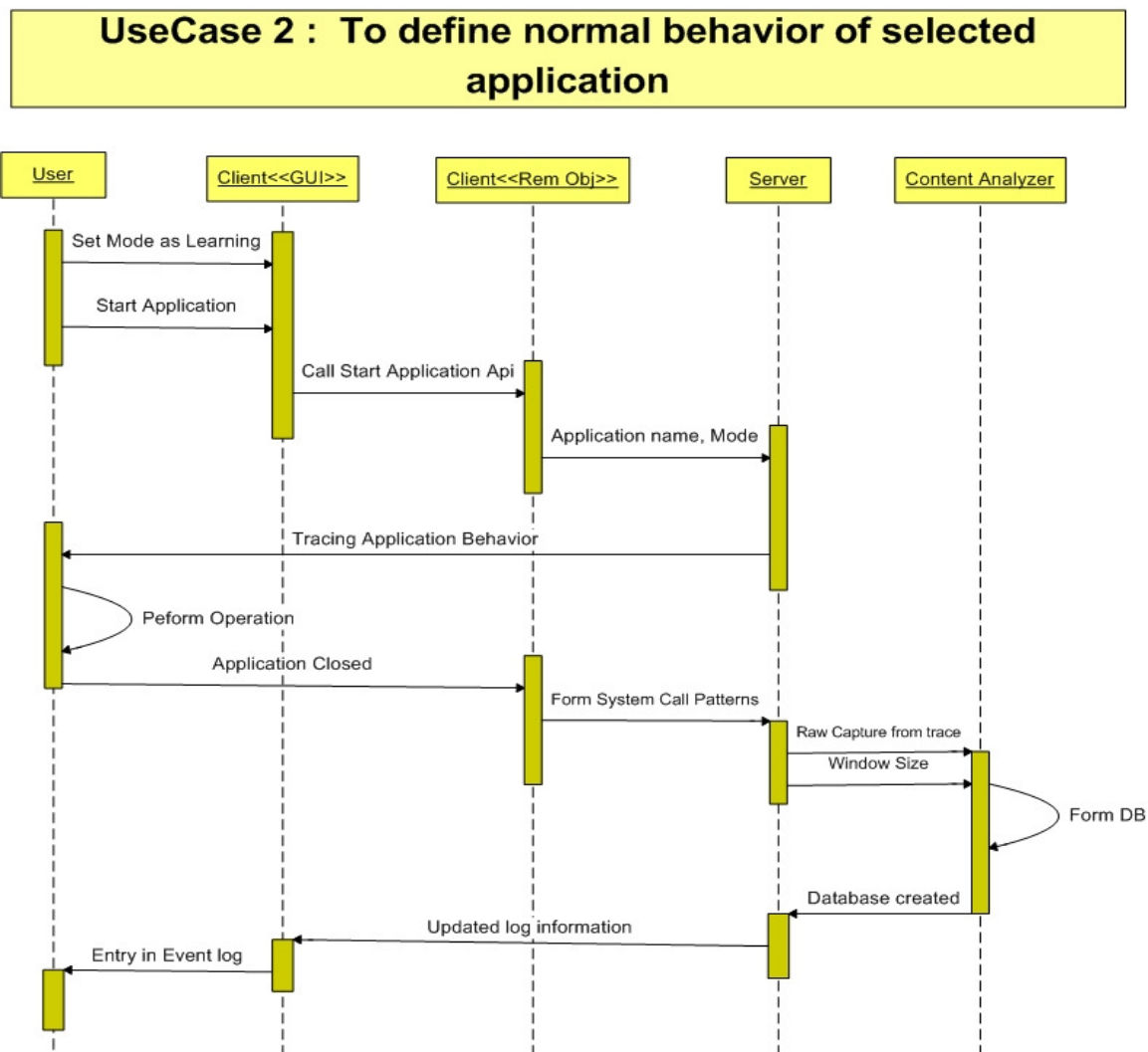


Figure 7 : Sequence Diagram-II

5.6 Use Case 2 : Sequence Diagram Description

The major components taking part during this use case are User from utilization side and Client (GUI) , remoting object of client, Server (listening requests) and Content Analyzer from the system side. The message passing among the components is visible from the above diagram. Though the use case shows a big deviation from the first case but internally it can be seen that there is no big difference from the implementation point of view.

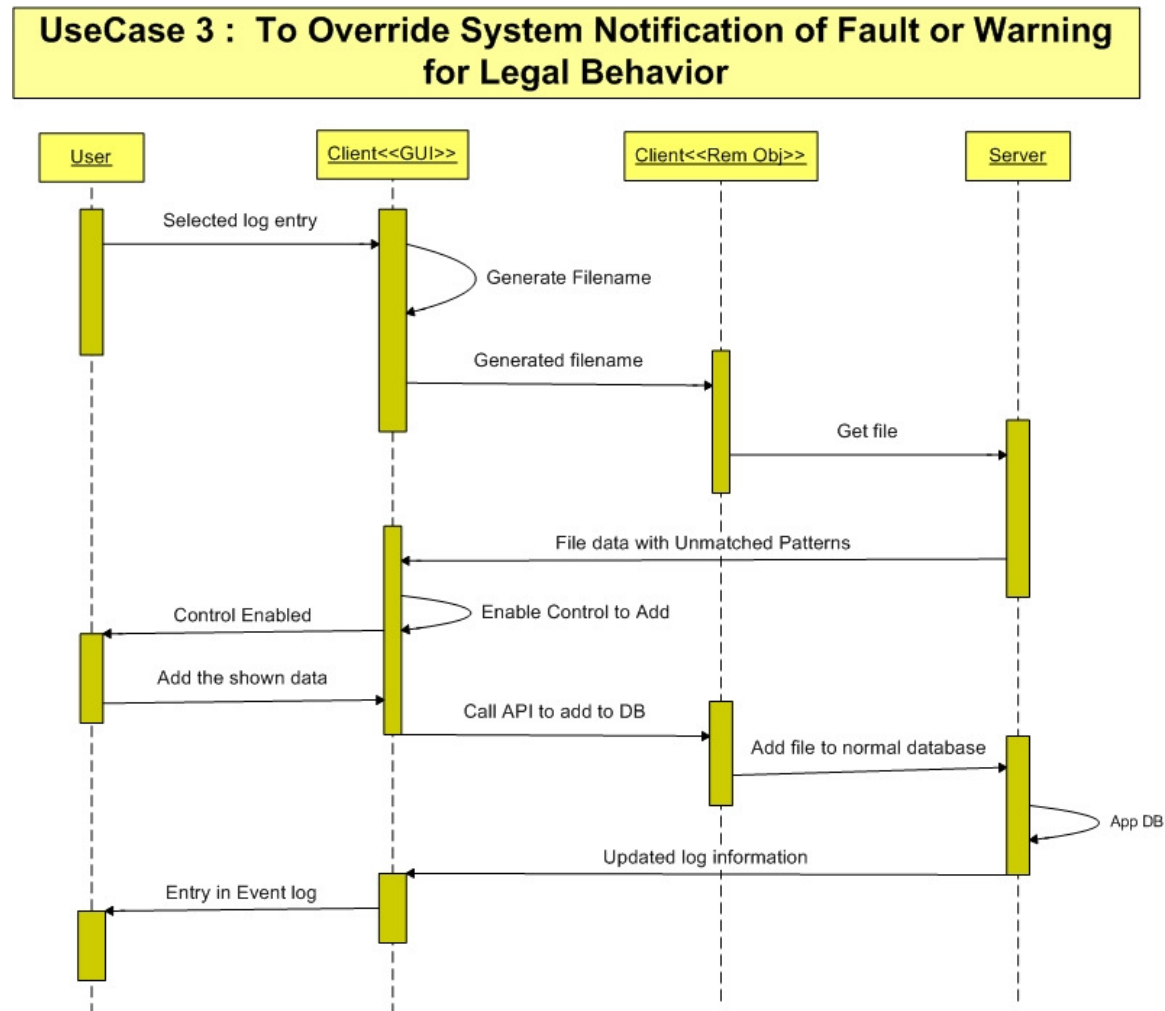


Figure 8 : Sequence Diagram-III

5.7 Use Case 3 : Sequence Diagram Description

The major components taking part during this use case are User from utilization side and Client (GUI), remoting object of client, Server (listening requests) and Content Analyzer from the system side.

5.8 Activity Diagrams

The activity diagram for the notification module has the three swim lanes based on the type of activity taking place – Requesting in which the client is making different requests to the client remoting object through GUI. The processing phase is one in which server is serving, processing client's request. Modules like content analyzer that act on the raw capture, analyzing application behavior etc. take place during this phase. The last – Notification is phase in which the server sends the notification to the user through GUI and also updates the various logs according to the action

Though the partitioning is to give better idea about the modules and in which phase they affect the system, not all the modules lie in single partition. Some modules actually can be thought of affecting two different phases of the system. These are seen to be intersecting two partitions in the activity diagrams. Notification action is done while the user runs an application in analysis mode and the system sends normal – fault or warning as notification type in the event log based on the analysis it performs. Defining application normal behavior is initial activity that user is expected to perform while building the normal database for the application.

Following next are the two activity diagrams for two major activities of the system– notification action and defining normal behavior of an application.

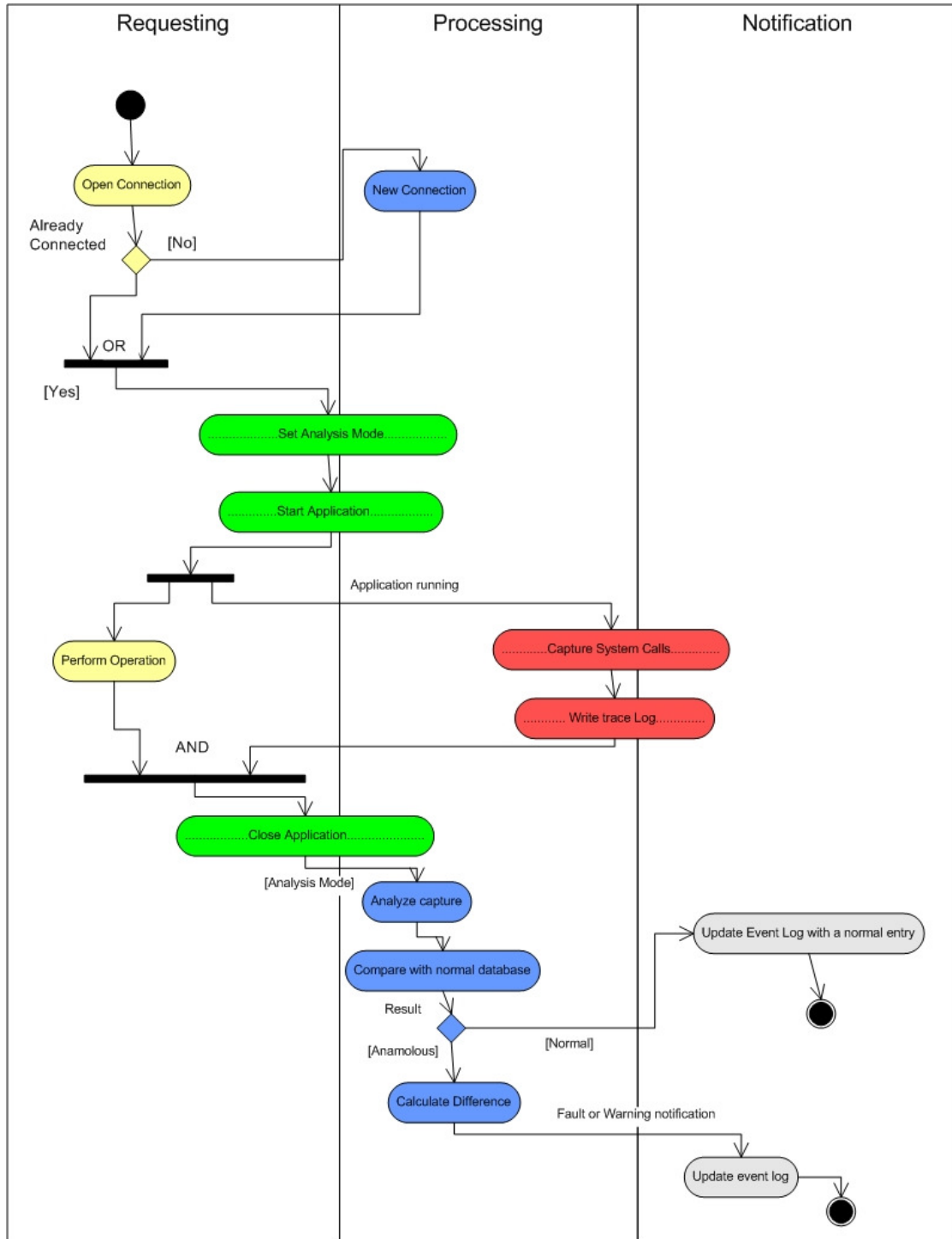


Figure 9 : Activity Diagram for Notification

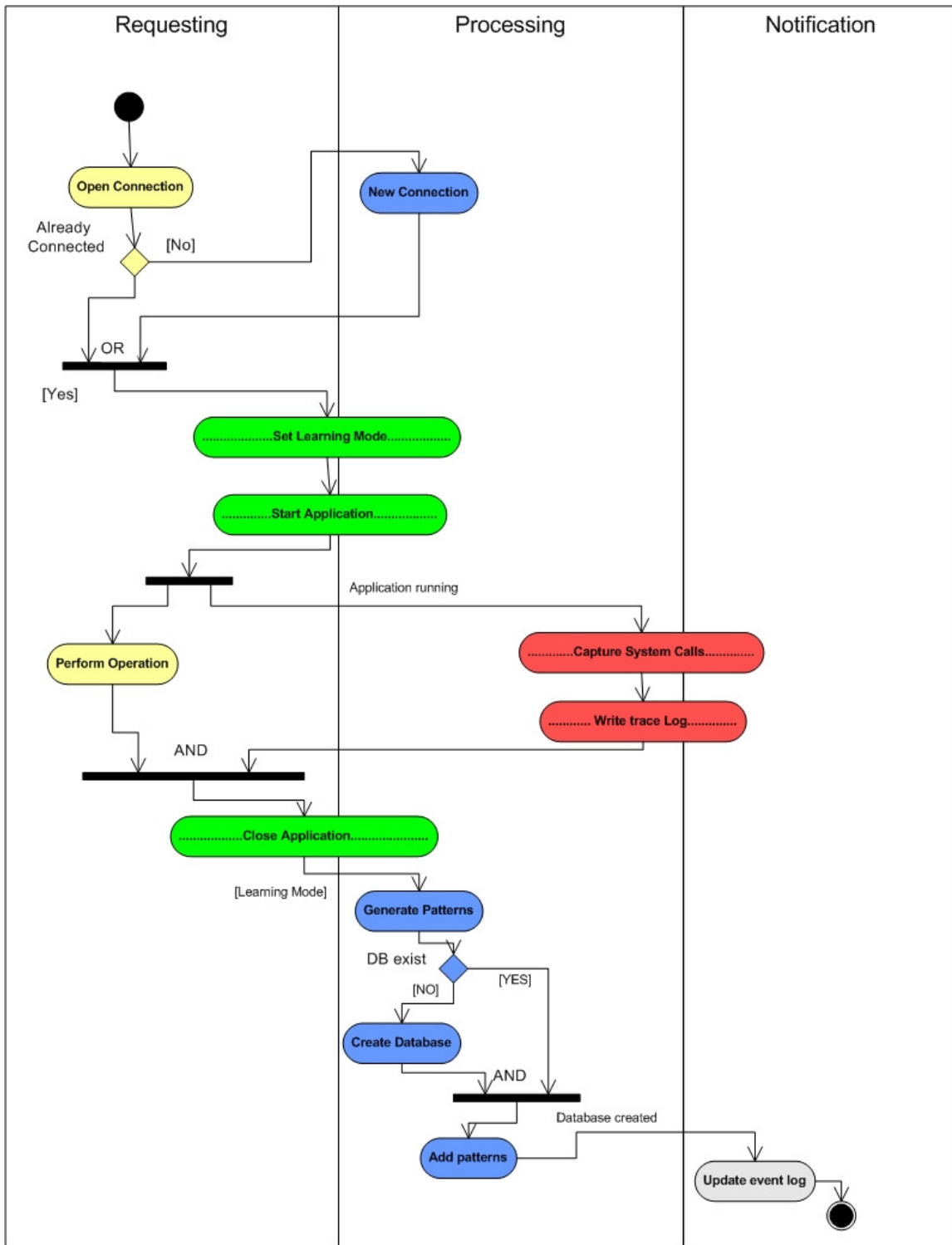


Figure 10 : Activity Diagram for Defining Normal Application Behavior

5.10 Description:

The class Form is one which would be containing major GUI components and CAppInfo is one that will have all the information about the current application with its process ids, time stamp etc. Form class can get list of active applications from the object ActiveApp. Also Form class interacts with the RemoteClient object to send the requests based on the events invoked by the user.

DetourDis is one of the components of Detour module – which is dealing with intercepting the system calls made by the application. It is using CImage and its derivatives to pass the application image – data – to trace.

Serializable and CVarVal are two major classes that enable logging of the traced system calls into the syslog file. The serialize and deserialize the data while storing and fetching by making a pipe connection with the application.

APICallAnalyzerRemote is the dealing with the calling of content analyzer module for content analysis of the data stored by the server. APICallAnalyzerHook is for hooking the client requests sent remotely to sever that actually implements CallAnalyzer module.

5.11 Data-Tier Design

The major data to be stored for the system can be classified in the following categories –

Logs – The record of user’s activity with the system and the analysis of the application behavior derived in the analysis mode.

Application system call patterns - These are the unique patterns that are created by the application that define the normal behavior of application.

Both these information are recorded in tabular format within the structured CSV files.

Following are the schema of the two major data storages.

5.11.1 Application behavior patterns

Timestamp	Thread Id	System Call Patterns
Date time format	Integer	String

Table 4: Schema for Patterns Db

5.11.2 Event Notification Log

Time	File	Status	Difference	Event	Description
Data Time	String	String	Float	String	String

Table 5: Schema for Event log file

For each application, we have its separate data which defines its normal behavior, while the event notification log is shared by the applications and is used to keep track of user’s activity with the system.

Chapter 6. Project Implementation

6.1 Serializable variant API parameter passing

AttachDLL.exe(executable part of this project) attaches detour DLL(NetSysApiMon.dll) to the launched application(like Outlook express or internet explorer or firefox) to intercept all the system calls along with their parameters. On NetSysApiMon.dll DLL process attach, all the BASM required system calls are detoured to collect the API names and their parameters. Detour DLL opens a named pipe connection to APICallAnalyzer server for passing application data. The timestamp of API call, process id, application info, thread id, API name and their parameters are sent to APICallAnalyzer server named pipe on every detoured system call. APICallAnalyzer server stores the API call information into CSV file for each application. This stored CSV file is used for analysis by the perl module.

Fig 13 below shows the sequence of actions on normal API calls without any detour. Fig 14 below shows the sequence of actions on API calls with detour. Fig 12 below shows the sequence of actions on application API calls with detour and APICallAnalyzer.

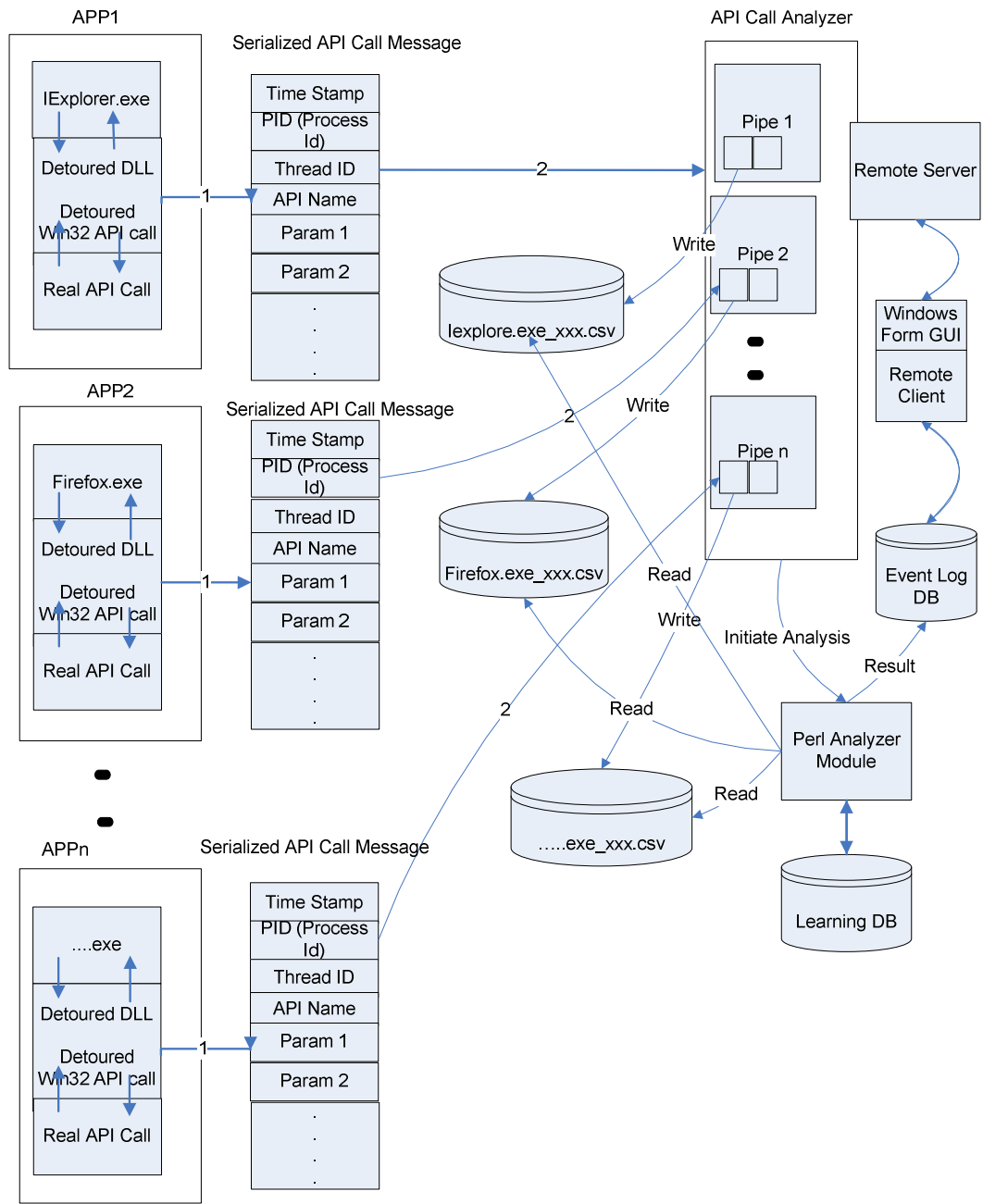


Figure 12 : Application, APICallAnalyzer Server with detour

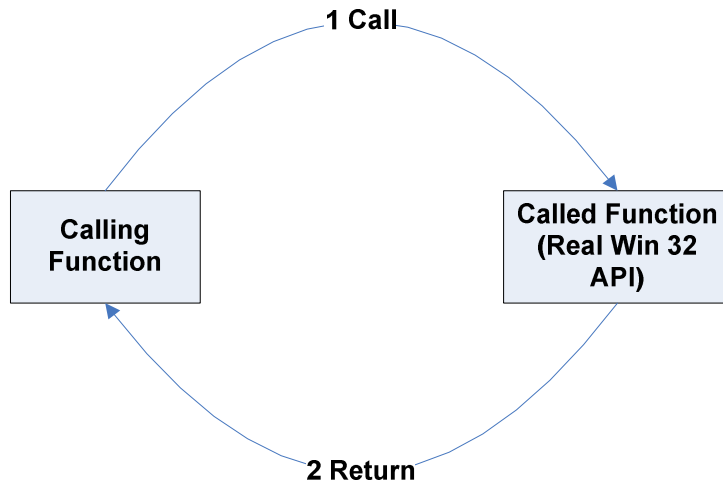


Figure 13: Application API Call without Detour.

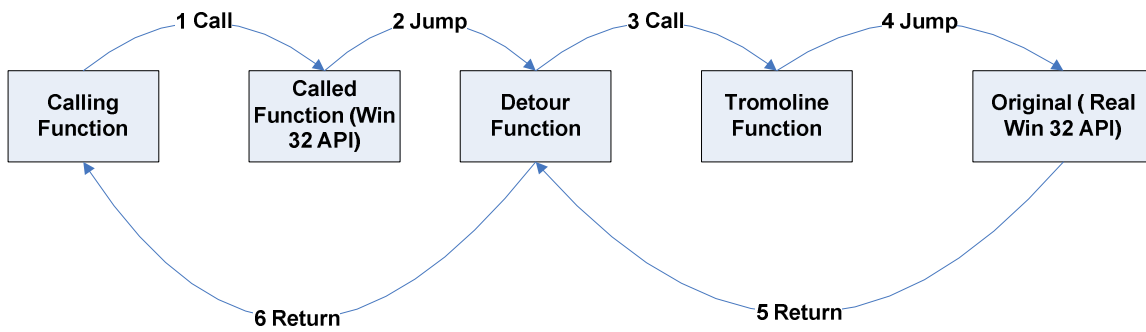


Figure 14: Application API Call with Detour

Passing process id, application info, thread id, API name and their parameters is not trivial. There are lots of restrictions, since code is executed in the context of different application. API Parameter types are not fixed. Parameters can be of type bool,int,char,char*,double etc., To handle different types of parameters, system creates serializable object with variant type parameters at the client side. Client's Serialized object is deserialized at the server side, and logged into application CSV log file. This application CSV log file is used for building learning database and also for analysis to determine the application behavior.

In the following section, we go over the implementation of CVarVal(Variant type class) and CSerializeParams(Serializable variant parameters class). These classes are very important for passing data from application APICallanalyzer module.

CVarVal(Variant type class)

CVarVal class allows storage of serializable variant parameter. Data types supported by CVarVal class are int, unsigned int, double, string and byte array(array of unsigned char). Various assignment operator function allows storing of values into variant type variable. Serialize() method serializes the variant type, length and value into byte stream. DeSerialize() method deserializes byte stream data into variant type. SaveToFile() method stores variant parameter value into CSV file.

Source Code with CVarVal implementation details:

```
////////////////////////////////////
// Custom variant class for storing different kinds data type(Similar to COM Variant)
////////////////////////////////////
#define MAX_VAR_PARAMS 10

typedef enum
{
    VVT_EMPTY = 0, // unknown type
    VVT_I4 = 1, // integer of size 4 bytes
    VVT_UI4 = 2, // unsigned integer of size 4 bytes
    VVT_DBL = 3, // size 8 bytes
    VVT_STR = 4, // ascii string
    VVT_BYTE_ARRAY = 5, // Byte Array
} VARIANT_TYPE;

////////////////////////////////////
//Variant to store API parameter of type int,double,string,byte array
////////////////////////////////////
class CVarVal
{
public:
    VARIANT_TYPE m_type; //Variant type
    UINT32 m_dataLen; //length of data
    BYTE *m_value; //variant value

public:
    //////////////////////////////////////
    //default constructor
    //////////////////////////////////////
    CVarVal();

    //////////////////////////////////////
    //destructor
    //////////////////////////////////////
    ~CVarVal();

    //////////////////////////////////////
    //Cleanup and free memory used for parameter storage
    //////////////////////////////////////
    void ClearMem();

    //////////////////////////////////////
    //Store byte array data as variant
    //////////////////////////////////////
    BOOL SetByteArray(BYTE *pByteAr, int nBytes)
    {
        ClearMem();
        m_type = VVT_BYTE_ARRAY;
        m_dataLen = nBytes;
        m_value = new BYTE[nBytes];
    }
};
```

```

        memcpy(m_value,pByteAr,m_dataLen);
        return true;
    }

    //////////////////////////////////////
    //assignment operator to allow copy of parameters for variant
    //////////////////////////////////////
    CVarVal& operator=(const CVarVal& varSrc)
    {
        if(this != &varSrc)
        {
            ClearMem();
            m_type = varSrc.m_type;
            m_dataLen = varSrc.m_dataLen;
            m_value = new BYTE[m_dataLen];
            memcpy(m_value,varSrc.m_value,m_dataLen);
        }
        return *this;
    }

    //////////////////////////////////////
    //assignment operator to allow copy of parameters for const char *
    //////////////////////////////////////
    CVarVal& operator=(PCSTR str);

    //////////////////////////////////////
    //assignment operator to allow copy of parameters for double
    //////////////////////////////////////
    CVarVal& operator=(double dblSrc);

    //////////////////////////////////////
    //assignment operator to allow copy of parameters for integer
    //////////////////////////////////////
    CVarVal& operator=(int intSrc);

    //////////////////////////////////////
    //assignment operator to allow copy of parameters for unsigned integer
    //////////////////////////////////////
    CVarVal& operator=(unsigned int intSrc);

    //////////////////////////////////////
    //Get the length(in byte count) required to store parameter
    //////////////////////////////////////
    int GetSize()
    {
        return m_dataLen+sizeof(m_type)+sizeof(m_dataLen);
    }

    //////////////////////////////////////
    //Serialize parameter by storing data type, length and value into byte stream
    //////////////////////////////////////
    BOOL Serialize(BYTE *pSerBuf, int nMaxSize)
    {

```

```

        if(nMaxSize < GetSize())
            return false;
        memcpy(&pSerBuf[0],&m_type,sizeof(m_type));
        memcpy(&pSerBuf[sizeof(m_type)],&m_dataLen,sizeof(m_dataLen));
        memcpy(&pSerBuf[sizeof(m_type)+sizeof(m_dataLen)],m_value,m_dataLen);
        return true;
    }

    ///////////////////////////////////////////////////////////////////
    //Deserialize parameter by extracting data type,length and value from the byte stream
    ///////////////////////////////////////////////////////////////////
    BOOL DeSerialize(BYTE *pSerBuf)
    {
        ClearMem();
        memcpy(&m_type,&pSerBuf[0],sizeof(m_type));
        memcpy(&m_dataLen,&pSerBuf[sizeof(m_type)],sizeof(m_dataLen));
        m_value = new BYTE[m_dataLen];
        memcpy(m_value,&pSerBuf[sizeof(m_type)+sizeof(m_dataLen)],m_dataLen);
        return true;
    }

    ///////////////////////////////////////////////////////////////////
    //print variant parameter info to console window for troubleshooting purpose
    ///////////////////////////////////////////////////////////////////
    void Print();

    ///////////////////////////////////////////////////////////////////
    //Save parameter to CSV file with Type and data
    ///////////////////////////////////////////////////////////////////
    BOOL SaveToFile(FILE *pFile)
    {
        char tempBuf[SYELOG_MAXIMUM_SERIALIZED_DATA]="";
        int iBufUsed=0;
        char byteData[4];
        if(!pFile)
            return false;

        switch(m_type)
        {
        case VVT_I4:
        case VVT_UI4:
            sprintf(tempBuf,"I4:%d",(int)*(int *)m_value);
            break;
        case VVT_DBL:
            sprintf(tempBuf,"DBL:%f",(double)*(double *)m_value);
            break;
        case VVT_STR:
            _snprintf_s(tempBuf,SYELOG_MAXIMUM_SERIALIZED_DATA,SYELOG_MAXIMUM_SERIALIZED_
            DATA-10,"STR:\\"%s\\", (PCSTR)m_value);
            break;
        case VVT_BYTE_ARRAY:
            sprintf(tempBuf,"BAR:\\"");
            iBufUsed = strlen(tempBuf);

```

```

10;i++)
    for(int i=0;i<m_dataLen && iBufUsed<SYELOG_MAXIMUM_SERIALIZED_DATA-
    {
        _snprintf_s(byteData,4,4,"%02x ",m_value[i]);
        strcat(tempBuf,byteData);
        iBufUsed += 3;
    }
    strcat(tempBuf,"\\");
default:
    break;
}

fwrite(tempBuf,sizeof(BYTE),strlen(tempBuf),pFile);
return true;
}
};

```

CSerializeParams(Serializable Variant parameters class)

CSerializeParams class allows any number of variant parameters to be stored. SetParam() method allows to store variant param(CVarVal) object at the specified index. GetParam() method can be used to retrieve the variant param by index. Serialize() method serializes all the variant parameters into stream of bytes. SaveToFile() method stores all the variant parameters into CSV file. CSerializeParams(int nMaxParams) constructor is used to store the variant params from the detour function. CSerializeParams(BYTE *pSerBuf, int nMaxSize) is used to deserialize variant params from byte stream on APICallAnalyzer server side.

Source Code with CSerializeParams implementation details:

```
////////////////////////////////////
//Serialize/Deserialize parameter(Used for serializing/deserializing API parameters)
////////////////////////////////////
class CSerializeParams
{
protected:
    CVarVal *m_pVarValParams;
    int    m_nParams;

public:
    //////////////////////////////////////
    // constructor for creating Serializable parameters
    //////////////////////////////////////
    CSerializeParams(int nMaxParams=10)
    {
        m_nParams = nMaxParams;
        if(m_nParams > MAX_VAR_PARAMS)
            m_nParams = MAX_VAR_PARAMS;
        m_pVarValParams = new CVarVal[m_nParams];
    }

    //////////////////////////////////////
    // constructor for deserializing parameters from byte stream
    //////////////////////////////////////
    CSerializeParams(BYTE *pSerBuf, int nMaxSize)
    {
        memcpy(&m_nParams,&pSerBuf[0],sizeof(m_nParams));
        if(m_nParams > MAX_VAR_PARAMS)
            m_nParams = MAX_VAR_PARAMS;
        m_pVarValParams = new CVarVal[m_nParams];

        int iCurSize = sizeof(m_nParams);
        for(int i=0;i<m_nParams;i++)
        {
            m_pVarValParams[i].Deserialize(&pSerBuf[iCurSize]);
            iCurSize += m_pVarValParams[i].GetSize();
        }
    }

    //////////////////////////////////////
    // Set the parameter Value for the given index
    //////////////////////////////////////
    BOOL SetParam(int paramIdx, const CVarVal &varVal)
    {
        if(paramIdx < 1 || paramIdx > m_nParams)
            return false;

        m_pVarValParams[paramIdx-1]=varVal;
        return true;
    }
}
```

```

////////////////////////////////////
// Get the parameter value for the given index
////////////////////////////////////
BOOL GetParam(int paramIdx, CVarVal &varVal)
{
    if(paramIdx < 1 || paramIdx > m_nParams)
        return false;

    varVal = m_pVarValParams[paramIdx-1];
    return true;
}

////////////////////////////////////
// Return number of bytes required to serialize all the parameters
////////////////////////////////////
int GetSerializeBufSize()
{
    int iSize = 0;
    for(int i=0;i<m_nParams;i++)
    {
        iSize += m_pVarValParams[i].GetSize();
    }
    return iSize+sizeof(m_nParams);
}

////////////////////////////////////
// Serialize the all parameters into stream of bytes
////////////////////////////////////
BOOL Serialize(BYTE *pSerBuf, int nMaxSize)
{
    int iSize = GetSerializeBufSize();
    if(nMaxSize < iSize)
        return false;

    memcpy(pSerBuf,&m_nParams,sizeof(m_nParams));
    int iCurSize = sizeof(int);
    for(int i=0;i<m_nParams;i++)
    {
        m_pVarValParams[i].Serialize(&pSerBuf[iCurSize],m_pVarValParams[i].GetSize());
        iCurSize += m_pVarValParams[i].GetSize();
    }
    return true;
}

////////////////////////////////////
// Print all the parameters info to console window
////////////////////////////////////
void Print();

////////////////////////////////////
// Save API info in following CSV format:
//      Time,Thread Id, API Name, Param1, Param2,...
////////////////////////////////////
BOOL SaveToFile(FILE *pFile, PCSTR pTimeStr, DWORD nTaskId)
{

```

```

char tempBuf[MAX_PATH]="";
if(!pFile)
    return false;

sprintf(tempBuf,"%s,%u",pTimeStr,nTaskId);
fwrite(tempBuf,sizeof(BYTE),strlen(tempBuf),pFile);
for(int i=0;i<m_nParams;i++)
{
    m_pVarValParams[i].SaveToFile(pFile);
    if(i != m_nParams-1)
    {
        strcpy(tempBuf," ");
        fwrite(tempBuf,sizeof(BYTE),strlen(tempBuf),pFile);
    }
}
strcpy(tempBuf,"\r\n");
fwrite(tempBuf,sizeof(BYTE),strlen(tempBuf),pFile);
return true;
}
};
////////////////////////////////////

```

6.1.1 APICallAnalyzer (Server)

APICallAnalyzer runs as a server and creates the named pipe connection to every application that is being detoured. APICallAnalyzer allows same application to run multiple instances at the same time. Process id is used as unique identifier for tracking active application instead of application name.

Typical operations performed for each application by APICallAnalyzer server are:

- Open CSV Log File – On application start
- Log to CSV file on API call – On win32 system API call
- Close CSV File – On application close
- Initiate Analysis or initiate adding to learning DB – After application close

RemoteServer implements .NET remoting interface by providing service for remote clients.

Following fig 15. shows the communication path between Remote Client(GUI) and Remote Server(APICallAnalyzer).

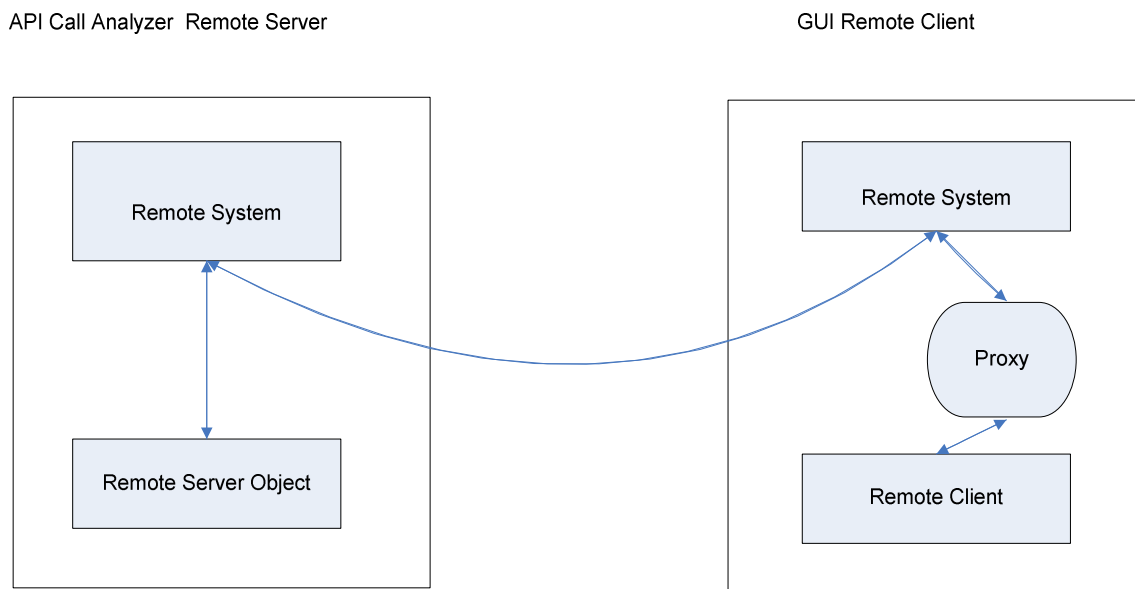


Figure 15: Communication - GUI Remote Client/Server (APICallAnalyzer)

Source code with of Remote Server interface:

```
////////////////////////////////////
// Remote Server implements interface for Remote Client(GUI)
////////////////////////////////////
namespace RemoteServer {

public ref class ApiCallAnalyzerRemote: public MarshalByRefObject
{
public:
    ApiCallAnalyzerRemote()
    {

    }

    //////////////////////////////////////
    // whether Client and Server are communicating or not
    //////////////////////////////////////
    bool IsConnected()
    {
        return true;
    }

    //////////////////////////////////////
    // Start Application in Analysis or Learning mode for API call data collection
    //////////////////////////////////////
    bool StartApplication(String ^appName, bool bAnalysisMode)
    {
        if(g_pAPIAnlHook)
            return g_pAPIAnlHook->StartApplication(appName,bAnalysisMode);
        return false;
    }

    //////////////////////////////////////
    // Get the list of currently active applications started using StartApplicaiton method
    //////////////////////////////////////
    String ^GetActiveApplications()
    {
        if(g_pAPIAnlHook)
            return g_pAPIAnlHook->GetActiveApplications();
        return "";
    }

    //////////////////////////////////////
    // Allows manual analysis of already existing log file
    //////////////////////////////////////
    bool StartManualAPIAnalysis(String ^appLogName)
    {
        if(g_pAPIAnlHook)
            return g_pAPIAnlHook->StartManualAPIAnalysis(appLogName);
        return false;
    }
}
```

```

        ////////////////////////////////////////////////////////////////////
        // Add Diff data to training or Learning database
        ////////////////////////////////////////////////////////////////////
bool AddApiCallSeqToTrainingData(String ^fileName)
{
    if(g_pAPIAnlHook)
        return g_pAPIAnlHook->AddApiCallSeqToTrainingData(fileName);
    return false;
}

        ////////////////////////////////////////////////////////////////////
        // Get list of currently completed applications
        ////////////////////////////////////////////////////////////////////
int GetLogCount()
{
    if(g_pAPIAnlHook)
        return g_pAPIAnlHook->GetLogCount();
    return 0;
}

        ////////////////////////////////////////////////////////////////////
        // Cleanup temp files used during analysis
        ////////////////////////////////////////////////////////////////////
int CleanUpTempFiles(int iParam)
{
    if(g_pAPIAnlHook)
        return g_pAPIAnlHook->CleanUpTempFiles(iParam);
    return 0;
}

        ////////////////////////////////////////////////////////////////////
        // Set Warning and Fault limit percentage for API call sequence difference
        ////////////////////////////////////////////////////////////////////
bool SetAnalysisParams(int iWarnLimit, int iFaultLimit)
{
    if(g_pAPIAnlHook)
    {
        g_pAPIAnlHook->m_iAnalysisWarnLimit = iWarnLimit;
        g_pAPIAnlHook->m_iAnalysisFaultLimit = iFaultLimit;
        return true;
    }
    return false;
}
};
}

```

6.2 GUI Implementation details

GUI implementation uses List View, Text Area and Data Grid View components of windows Forms. List View displays the currently active applications, Text Area displays log files, and Data Grid area displays the application API call analysis events. GroupBox component is used to display Status and Application Mode.

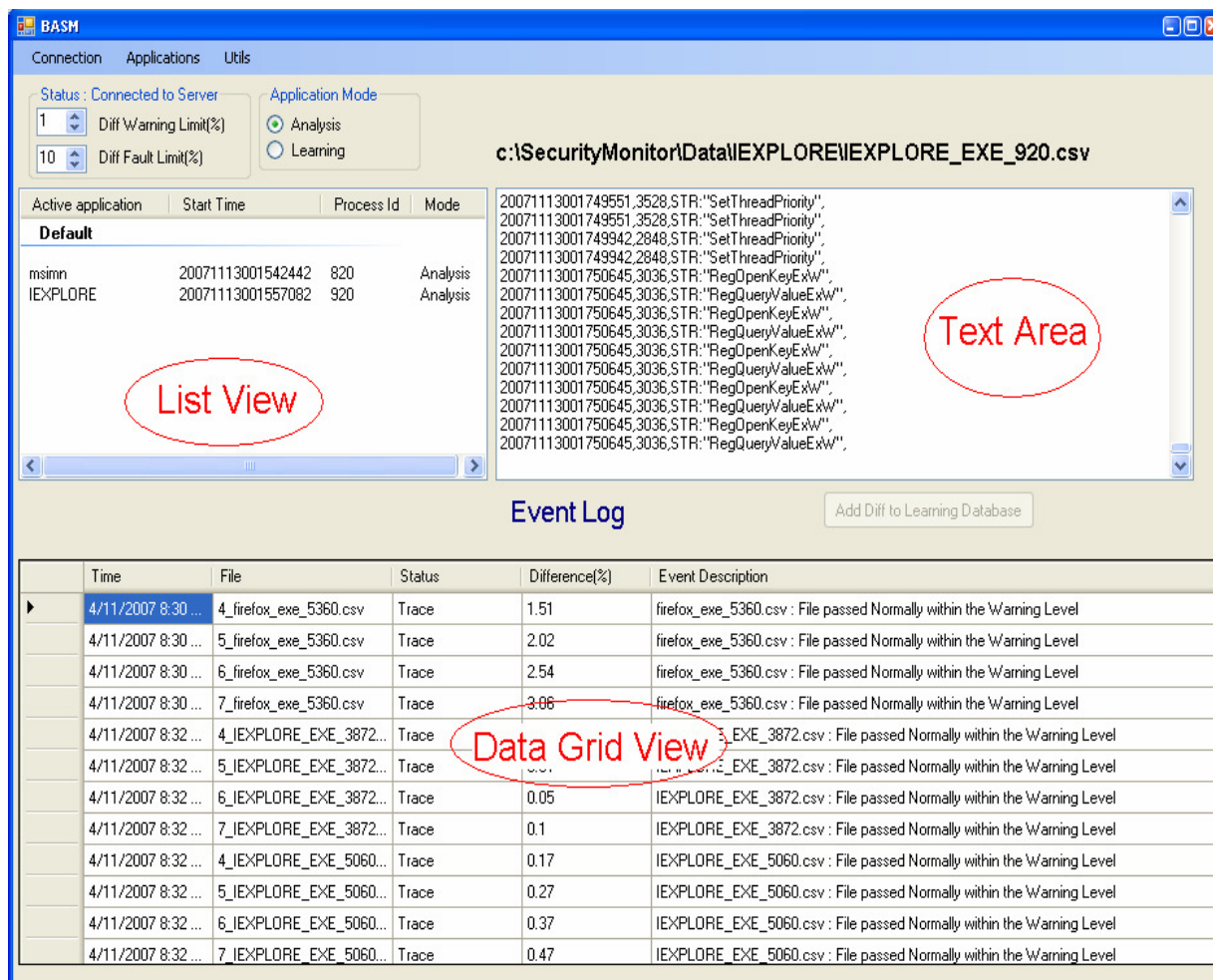


Figure 16: GUI component

BackgroundWorker component is used for updating Data Grid View data asynchronously and Text Area data. GUI uses .Net Remoting interface to communicate to APICallAnalyzer server.

6.3 The Analyzer Module

Depending on the mode set in the GUI, the output capture logged into either “\raw_capture” or “\create_training” directory. If the mode is in the learning mode which means that the Normal Behavior is on the learning mode. On the other hand, if the mode set in the GUI is Analysis mode, the system calls are logged on “raw_capture”. The application that caused the system call log is stored as a .csv file with the specific processID.

6.3.1 Training Mode

Once the system calls are captured, the name of the file will decide the Normal Train to add to. Every subsequent capture in the Training Mode will be added to its corresponding Normal train. The algorithm was defined in a way that each system calls will be grouped in a sequence of (4, 5, 6, 7) which we call it as a window size. For each logged system calls in the create_training directory, there will be 4 different training files generated and will be added to the Normal train for its corresponding window size. The perl files associated with each “Training Mode” is create_training.pl and myglobal.pl. Below is the explanation of how the Normal Behavior is created.

As soon as application is started from the GUI, the system calls, time and ThreadID's are logged. At the termination of the application, the file will be unlocked and further analysis can hence be performed on that file. The create_training.pl is kicked off at regular intervals to check if there are any files in that directory. If any files found, start processing them to add them to the Normal Behavior sequence calls.

Let us assume here, the application was Out Look Express. The csv generated would be something like msimn_exe_3342.csv. This means that the applications is "msimn" (Outlook Express). 3342 is the process id for that particular application. Also if there is another application logged like explorer, it would be iexplore_exe_4432.csv

The file has three main elements captured – ThreadID, System Call, Timestamp. Each ThreadID generates a big set of series of system calls. Each series is considered as an input to the Normal Training. The duplicates are removed from the normal training.

6.3.2 Analysis Mode

The Analysis mode can be run in two scenarios. One that there is a training database that defines a normal behavior and second that is captured henceforth which will be an addition to the Normal behavior train. Once the capture is found in the raw_capture directory, the parse.pl which is called at a regular interval from the GUI, processes each and every file from the capture. The Normal train is decided by the application name that the capture was taken from. The analysis is basically done by comparing the set of sequence calls from the normal train. The capture is processes using the threadID as one set of sequence calls. This

set of sequence calls are grouped on the window size ranging from 4 to 7. Each window is compared to that specific set of sequence of system calls from the corresponding “window sized train” and if that behavior sequence of system calls was not found, it will be registered as different from the normal behavior.

Here is an example of 4-7 set of sequence calls

4 : CreateFileW,CreateFileW,CreateFileW,RegOpenKeyExW

5 : htonl,htonl,CreateWindowExW,CreateFileW,send

6 : htonl,htonl,listen,LoadLibraryA,LoadLibraryExW,htonl

7 : htonl,listen,LoadLibraryA,LoadLibraryExW,htonl,htonl,Connect

and so on..

Once the difference is found for that particular capture the file that is different (only the differenced part) is saved in the “diff” directory. Each file creates 4 different files of either “diff” or “passed_OK” directory. If no difference was found for that window sized capture, it will be placed under “passed_OK” directory. Every file created in any of the two directory will be will be logged. A percentage difference will be logged. This percentage difference is compared to the original file (with window size x) with the difference found from the normal train (window size x). So, if the original capture was of 100 system calls (window size 4), difference found was 15 set of 4 system calls from the Normal Train, it will be 15% deviation from the Normal Training Behavior. This percentage deviation will / may be different for different window size. After certain number of successive sanity runs, this percentage deviation might drop down to almost zero for different trains. Again, as per our

analysis performed, the database of normal behavior of window size incremented will be more compared to lesser window size.

There is a level that we can set from the GUI that decides the warning (Alert) message to the user in the Log.csv file. Our program is so well designed, that depending upon the user's requirements and the threat to his system, he can set this level locally. Instead of just keeping 1 level as (Fault – No Fault), the program has set 2 points in the percentage that decides the warning level.

Each train is logged into the “analysis/X_application.csv”, where X is the window size of the list of system calls. Each csv file in that directory consists of the number of times the application was run against the normal train (compared for the difference), the number set of system calls different from the Normal Train, total number of system calls (grouped to the window size), the Train Size at that particular run, time stamp and the percentage deviation. A bunch of analysis can be done from that data gathered.

Once the analysis is performed, for each run there will be 4 different files generated for each run with one window size. In our program, we have set 4 different window sizes (4-7). So that leaves us with 16 files created for one analysis run. If the file has passed the Normal behavior, the file will be saved in “passed_OK” and if not the file differenced file will be saved under “/diff” directory. After the user / admin looks at the percentage deviation from the GUI logged in log.csv file, the user can either discard the file and call for clear log files, or the user can “add” the % diffed to that corresponding normal train. For eg. Consider

4_msimn_exe_443.csv is the file that was found in the “/diff” directory. The size of this file is found out to be 200 records. This means that the window size of the captured diff is ‘4’. Application used is Outlook Express and the process ID is 443. At this point the percentage deviation is already logged in log.csv and under “/training/analysis/4_msimn.csv”. When the user clicks on that particular diffed file from the GUI, and clicks on “add to database” the database Normal train for window size 4 and Outlook express is appended. The file gets disabled once the database already has an entry for that file.

6.4 Program Code Details

The Analyzer module has these many Perl Implementation files:

- *Init.pl*

This file sets up the initialization parameters. It creates certain directories which are required for analysis. It also cleans up “current.txt” that shows the current analyzes performed on that capture.

- *Create_training.pl*

This file tracks the normal behavior and considers it as a training mode. So whatever is done with the application, the set of system calls are tracked and saved for that particular window size that is set (4-7) in this function. We can also increase this depending on the security level we want to set. The duplicates are removed before they are entered into the train.

- *Parse.pl*

This is the main file that parses the capture and creates different files in “/diff/X_application_exe_processid” or under “/passed_OK/X_application_exe_processID”. This is created by parsing each file and each window size to its corresponding window sized application. The way it is done is quite interesting to know. The key feature to implement parsing here on each capture is based on “ThreadID”. Every multithreaded application has a set of thread that performs certain system calls. The timestamp of each system call will be different depending on the scheduler algorithm and the queuing mechanism for that particular call. So here we separate the system calls and create a series of array or each threadID. Each array has a sequence of that particular threadID. These system calls for each thread were slotted depending on the window size (eg. 4-7). Each slotted calls were compared to the calls that are found in the normal behavior defined until that time and if found a difference, that particular X sized window and creates the difference file with the X_application_exe_processID.csv, where X is the window size, which means that if a difference was found in the 4 point capture of Outlook Express, a file will be created in “/diff/” directory, with that file name. This csv flat file will include the set of 4 sequence calls that were not matched from the normal behavior compared with the train of 4_msimn.csv (normal behavior train with window size of 4). The parse.pl is pretty intelligent to decide the Normal behavior file that it has to compare with the particular application’s capture. For example, the once the slots are take for that threadID, depending upon the slot it will compare with slot_application. Slots of 4 window size for Outlook, will be compared with the 4 slots of outlook express and 5 with 5 and so on.

Concisely, each threadID is parsed and created a train of window size, compared to the corresponding window sized train and if found a difference, only the difference part is created in the file under “/diff/” and if whole capture is passed, only the file name is touched into the passed_OK directory and the messages are updated in the log.csv file. Update is also done in analysis directory. This analysis directory is almost similar to the Normal train, the only difference is that, each run has one entry that shows the current Normal Database size, the number of system calls that were different from the train and the total number of captured system calls. Percentage deviation is also calculated which decides the status message in Log.csv and is logged per run in analysis.csv.

- *Add_to_training.pl*

This file is called from the GUI when there is a difference found in the “diff” file , it is logged into the Log.csv. This log.csv is displayed with the level of deviation from the Normal Database Train. There is a description that shows about the file that were found in the diff directory. If the user decides that the behavior is still normal and he can then add that file to the Normal Behavior train. There is a button provided wherein he can select the file from the GUI and click on that button. Click on that button will call add_to_training.pl file. The duplicates have already been removed, so the only process to perform here is to append these files to the corresponding file of Normal Behavior. 4_msimn_exe_4444.csv when selected and the button ‘add to database’ clicked, will append the contents to 4_msimn.csv Normal Database train and so on. Next time when the same behavior occurs, will not be diffed, but will be found in the Normal train.

Hence, we will not be concerned with that particular difference in the behavior if it ever happens because it will be found in the normal behavior.

- *Cleanup.pl*

This file is called at the exit of BASM GUI. There are certain clean up required at the end of the application. If the argument passed to this is “files” it removes the dynamic log file "CURRENT.TXT" which appends every execution of analysis performed. It also removes any files in passed_OK directory. These files have only been touched and logged in the LOG.CSV. They are not required, but can also be kept for future requirement. This can be done by commenting this portion of the code. If the argument passed is “log” which means that we are clearing the LOG.CSV, this file gets cleared up.

- *Myglobal.pl*

This is the main file that has almost all the required global functions. Some of them might be re-used. Few functions exist in this file just for simplicity of the program code.

Here is a list of functions that exist and are explained

- `_dprint` : writes each executing step in the analysis into “current.txt”.
- `check_dir` : checks whether the directory exists or not.
- `Compare` : compares the captured run (slotted) with the window size (of that particular slot and same application). It also logs the result into the LOG.CSV and ANALYSIS.CSV.

- Create_train : this function creates the window size depending upon the input provided. We have set it to default as (4-7). It creates a train of those window and returns a reference of that array.
- Create_training : this function is intended for future use, wherein the use of time can be logged and can be used. It is not used currently, but we had created it before we implemented the threadID.
- Get_dir : returns the current working directory in a processed format.
- Get_files : a directory is passed and all the files present in that directory are returned as an array.
- Get_file_name_from_directory : this function give just the file name from the directory passed along with the file name. Eg. C:\SecurityMonitor\Perl_Module\diff\5_msimm_exe_4433.csv is passed and the return value of this function will be 5_msimm_exe_4433.csv.
- Get_time : returns the current time in human readable format to use in log and current.csv.
- Gprint : prints the results in LOG.CSV which is also defined globally. This file can also be changed.
- Readf : returns the array reference of the file passed as an argument.
- Remove_duplicate : This removes any duplicate in the file when either compared from normal train to the capture or when creating a normal database.
- Train_to_use : returns the train to use for comparison.

6.5 Test Cases Implementation

Testing had to be an important phase in the development of our application. The idea to trace the system calls of an application can lead to crash and abnormal behavior of an application. Hence we had to make sure that detouring of the system calls did not affect the functionality of the application.

Also all the desired system calls were being traced correctly with correct parameters being logged had to be checked. Hence we deployed 7 test cases that were covering all the different areas of the application. Following are the test cases implemented.

Tested By:	Venkatesh Babu	
Test Type:	Black Box	
Test Case Number:	TC1	
Test Case Name:	Detoured Outlook Express	
Test Case Description:	Testing functioning of detoured outlook express application	
Item(s) to be tested		
1.	Outlook express initialization	
2.	Send and Receive of mails	
Specifications		
Input		Expected Output
Application functions		Normal application behavior without any crash or problem
Procedural Steps		
1	Start the BASM application.	
2	Open connection to the server	
3	Click on Start Application and select Outlook express as application	
4	Outlook express should work without any problem.	
5	Send and Receive the mails which should get done successfully.	

Table 6: Test Case1 - Detoured Outlook Express

Tested By:	Venkatesh Babu	
Test Type:	Black Box	
Test Case Number:	TC2	
Test Case Name:	Detoured Web browsers	
Test Case Description:	Testing functioning of detoured web browser applications	
Item(s) to be tested		
1.	Internet Explorer initialization	
2.	Surfing on different websites – ww.youtube.com , www.yahoo.com	
3.	Firefox Initialization and browsing	
Specifications		
Input		Expected Output
Applications and websites		Normal application behavior without any crash or problem
Procedural Steps		
1	Start the BASM application.	
2	Open connection to the server	
3	Click on Start Application and select Internet Explorer as application	
4	It should work without any problem.	
5	Open web-site www.yahoo.com and www.youtube.com	
6	The sites should open without any problem	
7	Perform similar task but with Firefox as detoured application	

Table 7: Test Case2- Detoured Web browsers

This test case is important since we need to make sure that application does not hang or crash due to detouring of system calls.

Tested By:	Kunal Vyas	
Test Type:	White Box	
Test Case Number:	TC3	
Test Case Name:	Content Analyzer Algorithm Check	
Test Case Description:	Testing functioning of Algorithm to form patterns from Input raw capture of system calls	
Item(s) to be tested		
1.	Content Analyzer Algorithm and parser.pl	
2.	Uniqueness of patterns	
Specifications		
Input		Expected Output
CSV file with sequences of system calls		CSV file with unique patterns in group of 4,5,6,7 sequences of system calls
Procedural Steps		
1	Get a csv file of sequences of system calls.	
2	Feed it to the parser.pl module	
3	Check the output csv files	
4	Make sure there are four different csv files based on the sequences of window size.	
5	Make sure all the sequences of patterns are unique.	

Table 8: Test Case3- Content Analyzer Algorithm Check

Tested By:	Maulik
Test Type:	Black Box
Test Case Number:	TC4
Test Case Name:	Normal DB Creation
Test Case Description:	Testing functionality of adding patterns correctly for application behavior.
Item(s) to be tested	
1.	Content Analyzer Module add_training_data.pl
2.	Addition of new patterns to the csv file.
Specifications	
Input	Expected Output
Application behavior and several operations	CSV file with unique patterns in group of 4,5,6,7 sequences of system calls
Procedural Steps	
1	Start the BASM application.
2	Open connection to the server and set mode as learning.
3	Click on Start Application and select Outlook express as application
4	Perform operations
5	Check the appropriate csv file
6.	The csv file should contain patterns of system calls.

Table 9: Test Case 4- Normal Db Creation

Tested By:	Maulik
Test Type:	White Box
Test Case Number:	TC5
Test Case Name:	Validate Notification
Test Case Description:	Testing functionality of notification based on % differences found in the application behavior.
Item(s) to be tested	
1.	Content Analyzer Module parser.pl
2.	Algorithm to check fault and warning based on % difference
Specifications	
Input	Expected Output
Application behavior csv file.	Correct event notification based on % differences found.
Procedural Steps	
1.	Start the BASM application.
2.	Open connection to the server and set mode as analysis.
3.	Set appropriate Diff Warning limit and Diff fault limits
4.	Click on Start Application and select Outlook express as application
5.	Perform operations and close application
6.	Check the notification in Event log with % difference shown.
7.	Make sure the log displays correct notification based on difference limits set on the GUI.

Table 10: Test Case 5- Validate Notification

Tested By:	Kunal	
Test Type:	Black Box	
Test Case Number:	TC6	
Test Case Name:	Auto entry in Event log	
Test Case Description:	To test the functionality of auto updates in Event log as and when the event is done.	
Item(s) to be tested		
1.	Background worker thread	
2.	Event notification log.	
Specifications		
Input		Expected Output
Any Event		Correct event notification after some interval.
Procedural Steps		
1.	Start the BASM application.	
2.	Open connection to the server and set mode as analysis or learning	
3.	Set appropriate Diff Warning limit and Diff fault limits	
4.	Click on Start Application and select Outlook express as application	
5.	Perform operations and close application	
6.	Check the notification in Event log after some time and validate it has correct description based on the action and event performed.	

Table 11: Test Case 6 - Validate Auto Entry in Event Log

Tested By:	Kunal	
Test Type:	Black Box	
Test Case Number:	TC7	
Test Case Name:	Control Enabling	
Test Case Description:	To test the functionality enabling Add button on the GUI while the fault or warning type notification is selected in the event log.	
Item(s) to be tested		
1.	Add data to training set	
2.	Button control on the GUI	
Specifications		
Input		Expected Output
Warning or Fault event selection in the event log		Enabling of Add diff to training data button control on the GUI.
Procedural Steps		
1.	Start the BASM application.	
2.	Open connection to the server and set mode as analysis or learning	
3.	Select appropriate Fault or Warning type notification in the event log.	
4.	Check the button on GUI. It should be enabled.	
5.	Click on the add button.	
6.	The difference should be added to the normal database.	

Table 12: Test Case 7- Validate Control Enabling

Chapter 7. Performance and Benchmarks

BASM Analysis was performed for following three applications:

- Internet Explorer (iexplore.exe)
- Mozilla Firefox (firefox.exe)
- Outlook Express (msimn.exe)

7.1 Internet Explorer Analysis

- Analysis was performed by navigating different pages for <http://www.maulikthaker.com>
- Following 4 figures include analysis with window size 4 to 25 and up to 30 runs.
- Initially the Normal Train (Learning DB) was '0', that means the deviation for each train would be 100%.
- After each run, the system calls for normal behavior are tracked and logged.
- The figure below defines a signature of IE for the user surfing the website and this signature almost stabilizes at around 15 runs.

- No matter smaller the window size or larger, it takes approximately the same number of runs. Some of them might vary, but its obvious the greater the window size the better it gets. To obtain stability from Normal to Legal Behavior window size can be increased.

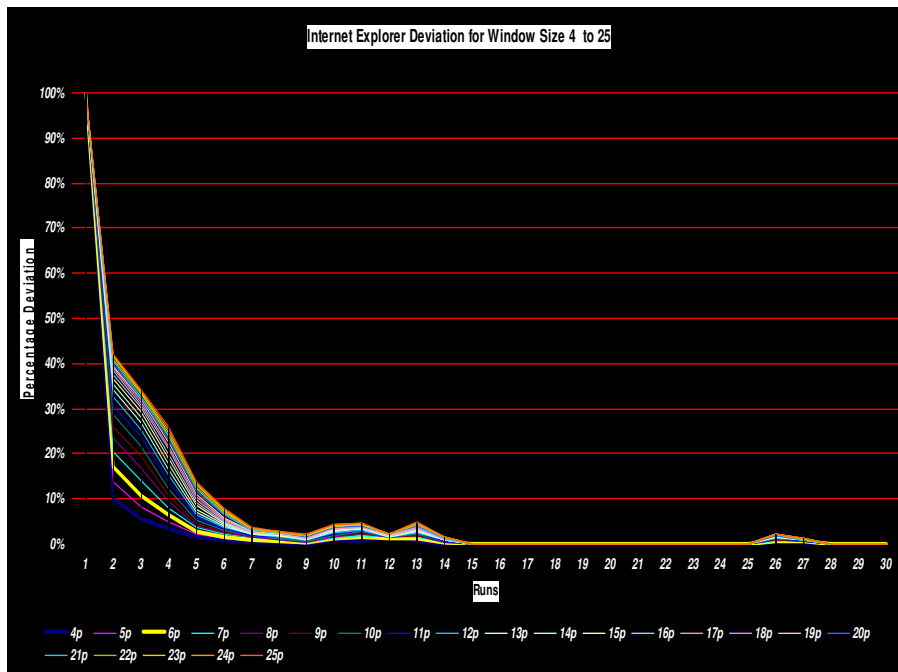


Figure 17: Chart – IE Deviation (window 4-25)

The graph below shows the database size that increases as the number of runs increases. When the growth of the window size stops after certain runs, the application is termed as legal application. In order to get a Normal Behavior, the graph should stabilize to be Behavior based. Looking at the graph, it is obvious that as the window size increases, certainly the database size increases. We have plotted here all the graphs of Database size v/s the number of runs for each window size.

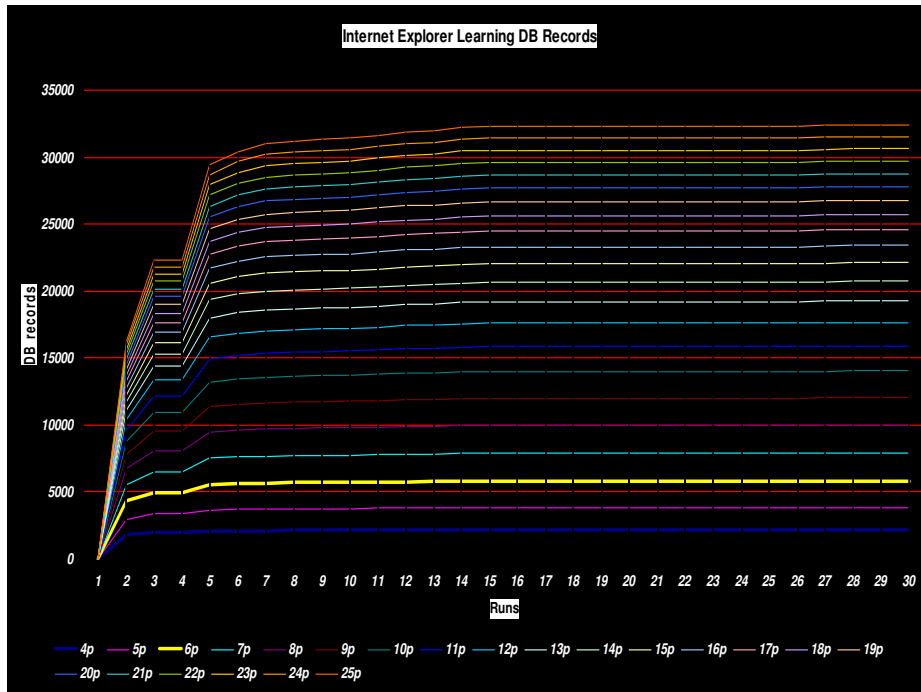


Figure 18: Chart – IE Learning DB (window 4-25)

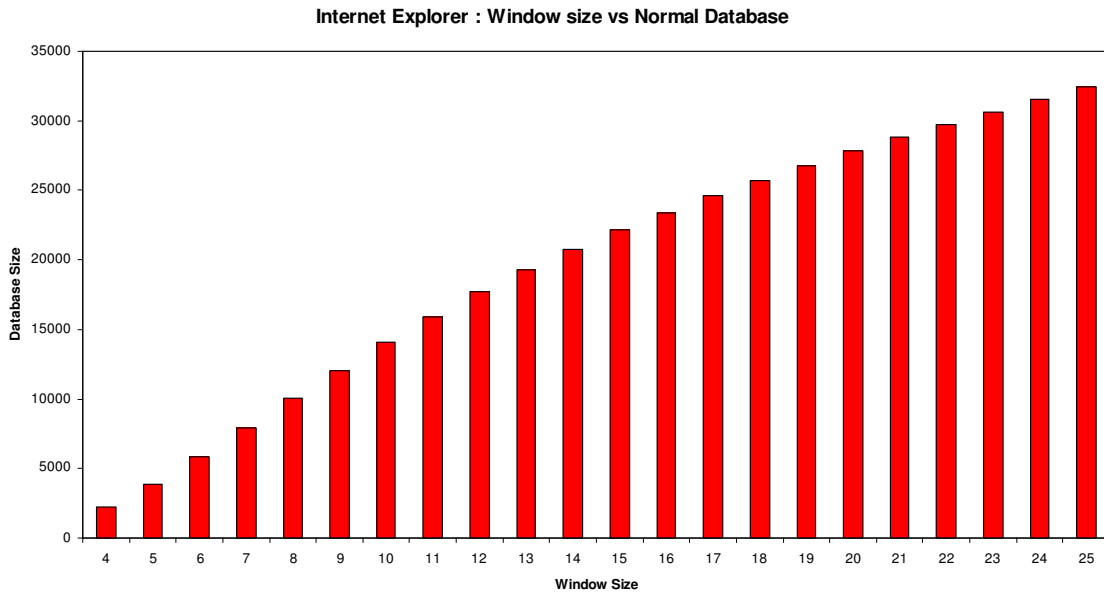


Figure 19: Chart – IE Window size vs Normal Database

Internet Explorer Deviations from SD

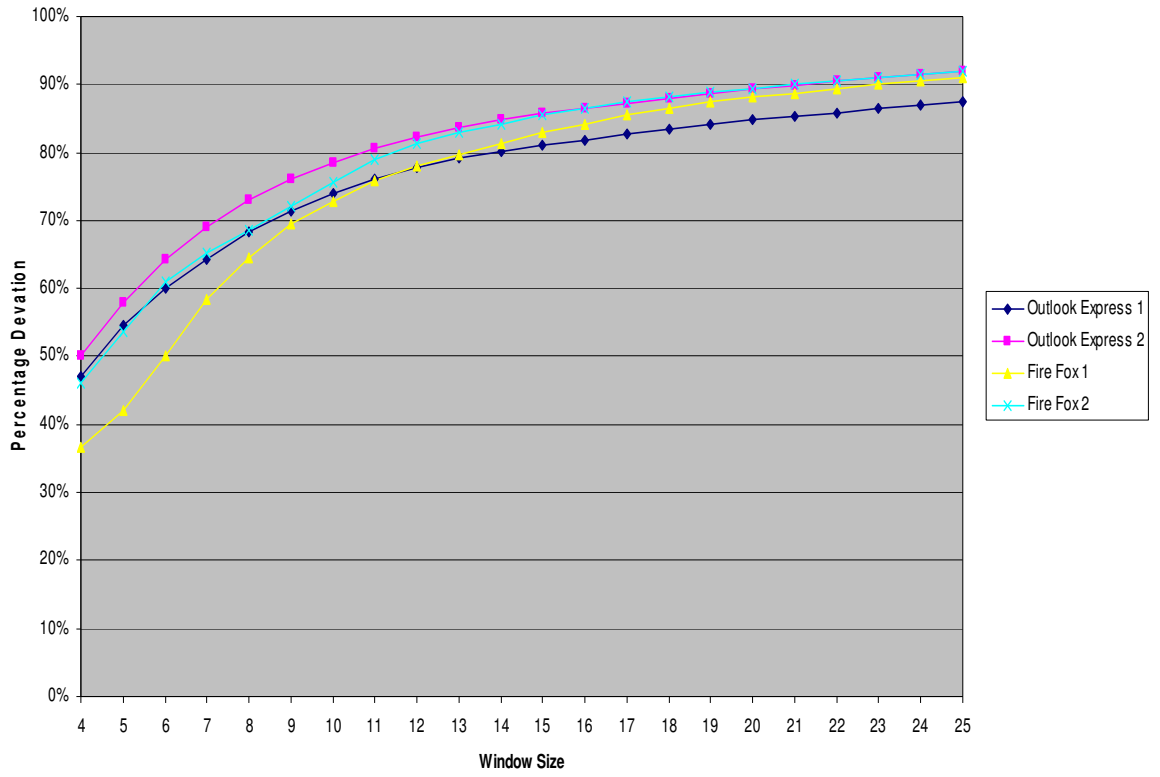


Figure 20: Chart – IE Percentage Deviation vs Window Size

7.2 Firefox Analysis

- Analysis was performed by navigating different pages for <http://www.maulikthaker.com>
- Following 4 figures include analysis with window size 4 to 25 and up to 30 runs.
- Initially the Normal Train (Learning DB) was '0', that means the deviation for each train would be 100%.
- After each run, the system calls for normal behavior are tracked and logged.
- The figure below defines a signature of FireFox for the user surfing the website and this signature almost stabilizes at around 27 runs.
- No matter smaller the window size or larger, it takes approximately the same number of runs. Some of them might vary, but its obvious the greater the window size the better it gets. To obtain stability from Normal to Legal Behavior window size can be increased.

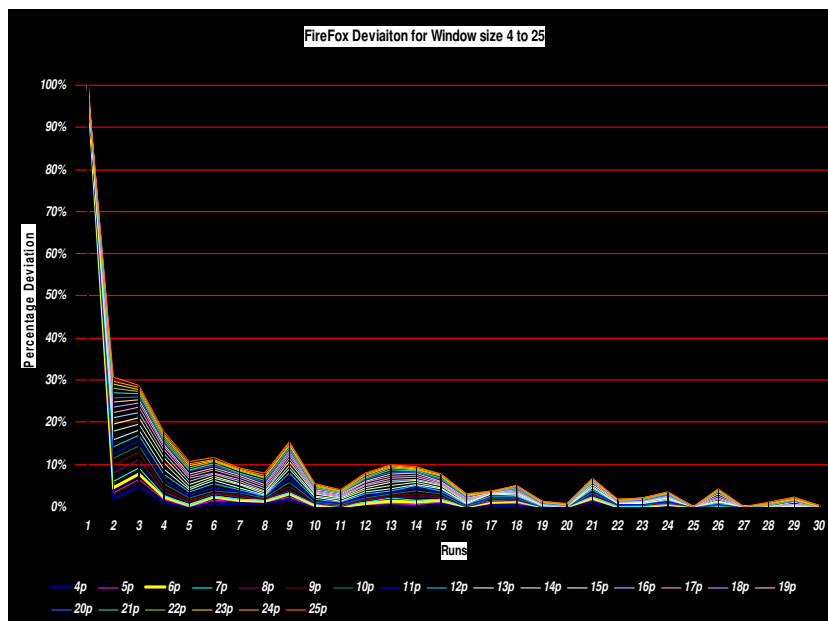


Figure 21: Chart – Fire Fox Deviation (window 4-25)

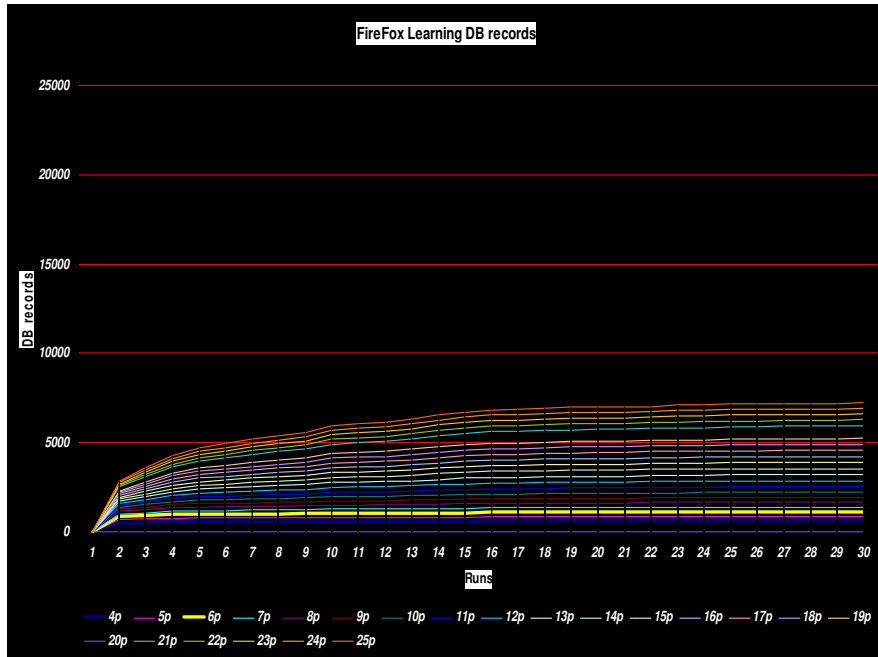


Figure 22: Chart – Fire Fox Learning DB (window 4-25)

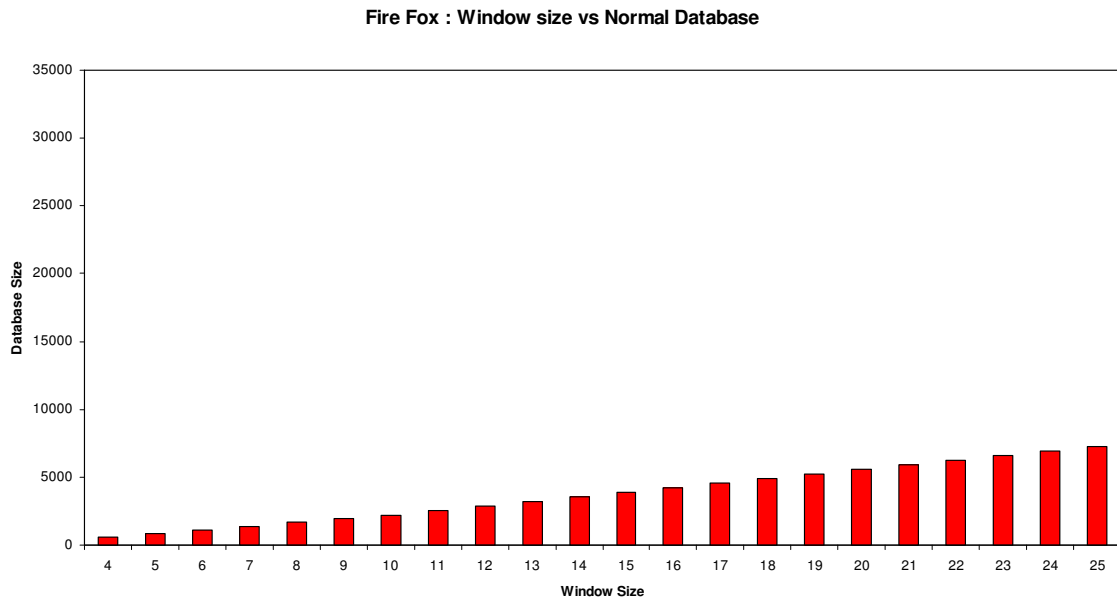


Figure 23: Chart – FireFox Window size vs Normal Database

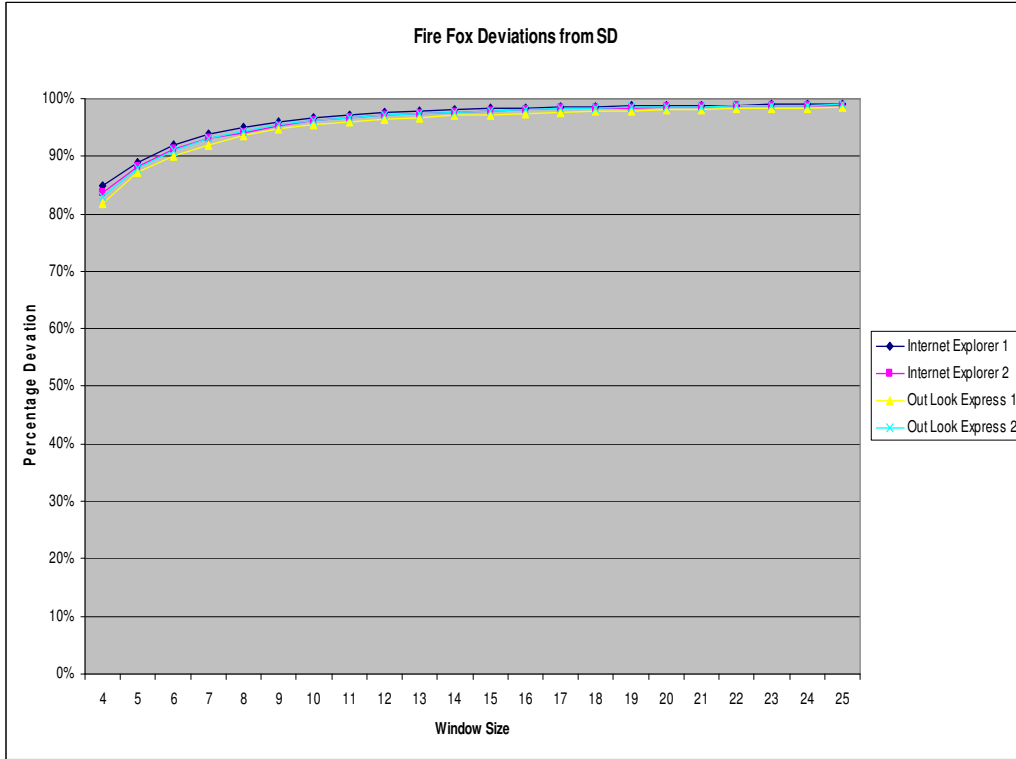


Figure 24: Chart – Fire Fox Percentage Deviation vs Window Size

7.3 Outlook Express Analysis

- Analysis was performed by performing the normal executions like Create Mail, Send/Receive, Reply, Forward, Reply All etc.
- Following 4 figures include analysis with window size 4 to 25 and upto 30 runs.
- Initially the Normal Train (Learning DB) was '0', that means the deviation for each train would be 100%.
- After each run, the system calls for normal behavior are tracked and logged.
- The figure below defines a signature of Outlook Express for the user surfing the website and this signature almost stabilizes at around 30 runs.
- No matter smaller the window size or larger, it takes approximately the same number of runs. Some of them might vary, but its obvious the greater the window size the better it gets. To obtain stability from Normal to Legal Behavior window size can be increased.

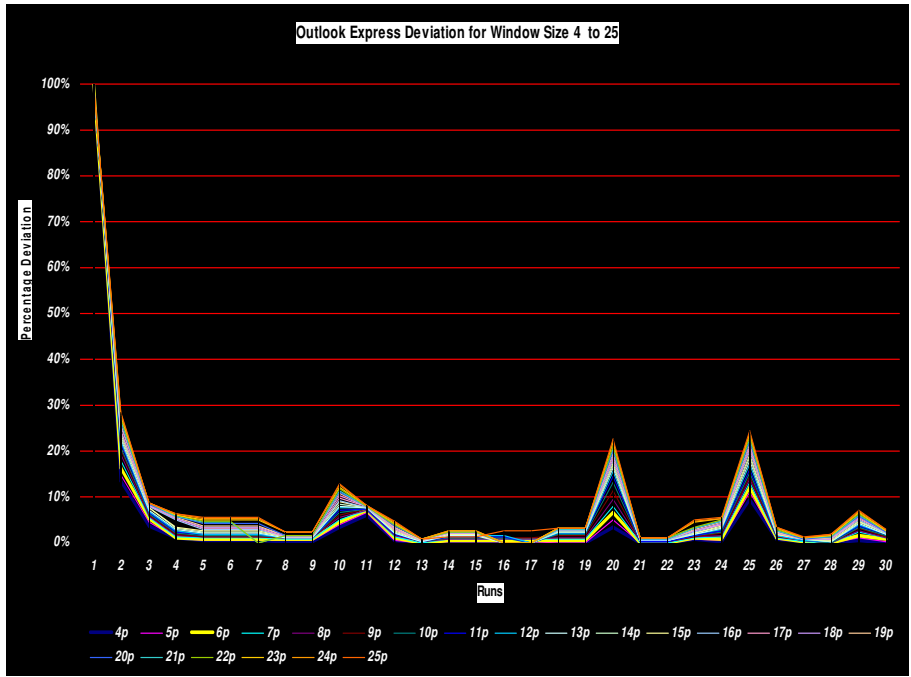


Figure 25: Chart – Outlook Express Deviation (window 4-25)

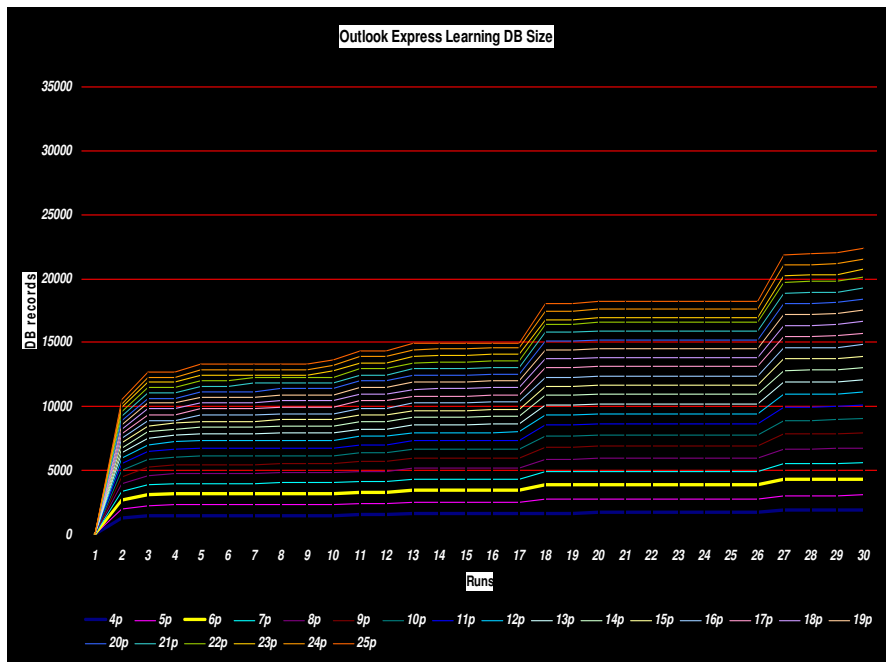


Figure 26: Chart – Outlook Express Learning DB (window 4-25)

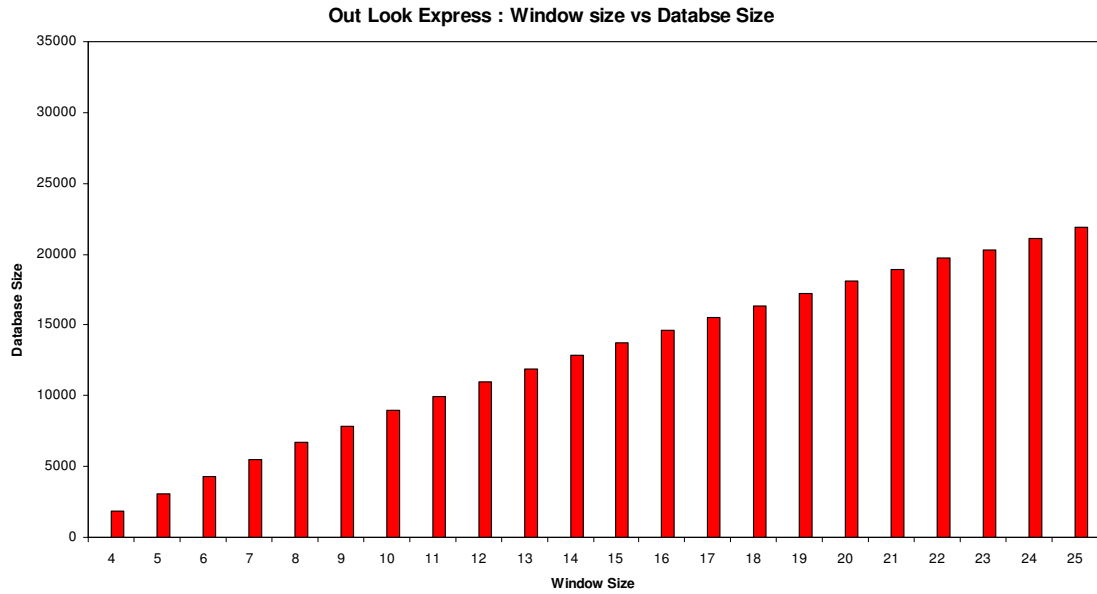


Figure 27: Chart – Outlook Express Window size vs Normal Database

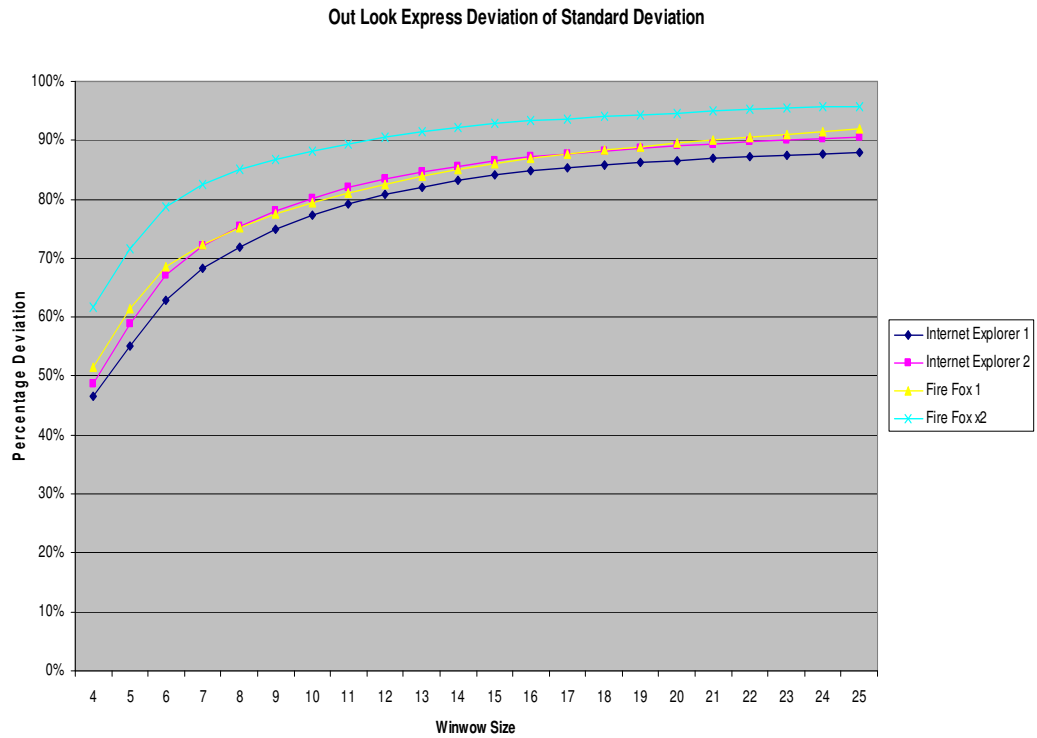


Figure 28: Chart – Outlook Express Percentage Deviation vs Window Size

Event Log					
Add Diff to Learning Database					
	Time	File	Status	Difference(%)	Event Description
▶	11/20/2007 6:58 PM	25_IEXPLORE_EXE_6712.csv	Fault	42	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:56 PM	24_IEXPLORE_EXE_6712.csv	Fault	41	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:54 PM	23_IEXPLORE_EXE_6712.csv	Fault	41	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:53 PM	22_IEXPLORE_EXE_6712.csv	Fault	40	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:52 PM	21_IEXPLORE_EXE_6712.csv	Fault	40	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:50 PM	20_IEXPLORE_EXE_6712.csv	Fault	40	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:49 PM	19_IEXPLORE_EXE_6712.csv	Fault	39	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:48 PM	18_IEXPLORE_EXE_6712.csv	Fault	38	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:47 PM	17_IEXPLORE_EXE_6712.csv	Fault	38	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:46 PM	16_IEXPLORE_EXE_6712.csv	Fault	37	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:46 PM	15_IEXPLORE_EXE_6712.csv	Fault	36	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:45 PM	14_IEXPLORE_EXE_6712.csv	Fault	35	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:44 PM	13_IEXPLORE_EXE_6712.csv	Fault	34	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:44 PM	12_IEXPLORE_EXE_6712.csv	Fault	32	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:43 PM	11_IEXPLORE_EXE_6712.csv	Fault	30	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:43 PM	10_IEXPLORE_EXE_6712.csv	Fault	28	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:43 PM	9_IEXPLORE_EXE_6712.csv	Fault	26	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:42 PM	8_IEXPLORE_EXE_6712.csv	Fault	23	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:42 PM	7_IEXPLORE_EXE_6712.csv	Fault	20	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:42 PM	6_IEXPLORE_EXE_6712.csv	Fault	16	IEXPLORE_EXE_6712.csv : File requires Immediate attention
	11/20/2007 6:42 PM	5_IEXPLORE_EXE_6712.csv	Fault	13	IEXPLORE_EXE_6712.csv : File requires Immediate attention
▶	11/20/2007 6:42 PM	4_IEXPLORE_EXE_6712.csv	Fault	10	IEXPLORE_EXE_6712.csv : File requires Immediate attention

Figure 29: Event Log (Each Run)

- We had 4 captures (2 IE and 2 FireFox). We analyzed these captures with Outlook Normal behavior. Same thing we repeated with FireFox Normal Behavior & IE Normal Behavior.
- For each window(4-25) we found a deviation.
- The figure below represents deviation found with different window sizes.
- This in a way helps us to decide, what window size to keep, beyond which efficiency does not increase or decrease.
- The longer sequences DOES NOT suffice good results.
- Moreover, as the number of sequence increases, time taken to calculate definitely increases.
- Next pictures shows the time difference of shorter sequence is within seconds compared to longer sequence which is in minutes

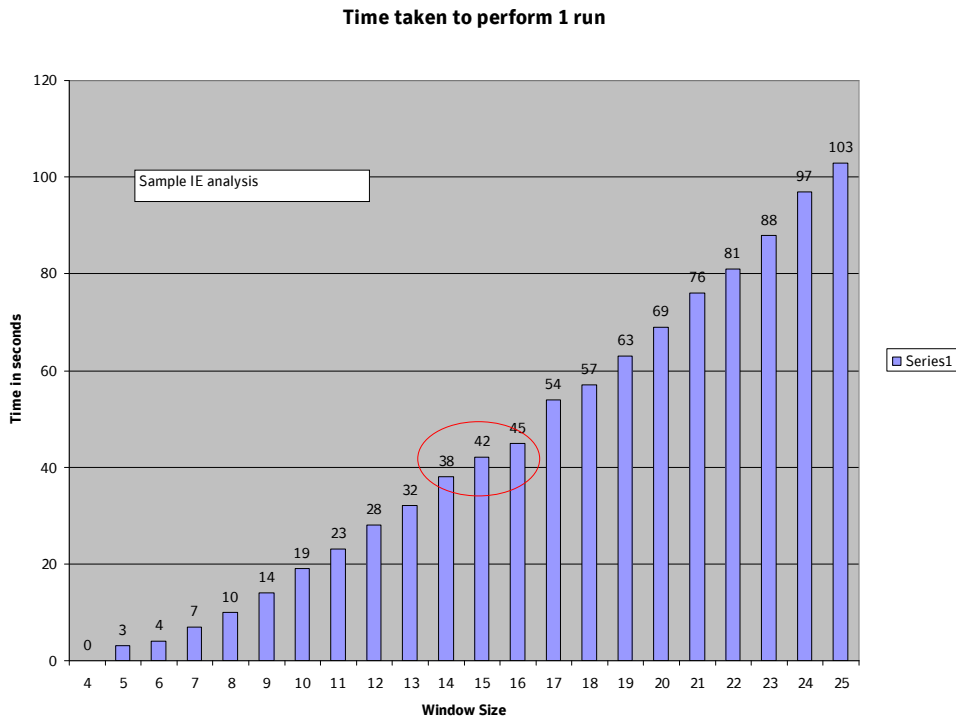


Figure 30: Chart – Time per run vs Window Size

A short sequence detailed analysis was performed on IE, and compared with the FireFox Capture. The 4 graphs below show each analysis performed.

- Its strange to know that the 0% deviation for window size ‘4’ requires less runs compared to as the window size increases.
- For 4 point capture, ~0% deviation reaches at 14th run, for 5 point capture, ~0% deviation reaches at 27th run, for 6 point capture, ~0% deviation reaches at 31th run, for 7 point capture, ~0% deviation reaches at 47th run.

- A drastic change in percentage deviation is when Fire Fox capture was compared to Internet explorer normal train behavior.
- This proves that any change in the behavior of train, irrespective of the same website being accessed, there is a change.
- Percentage change in Window size
 - 4 : 58%
 - 5 : 66%
 - 6 : 73%
 - 7 : 78%

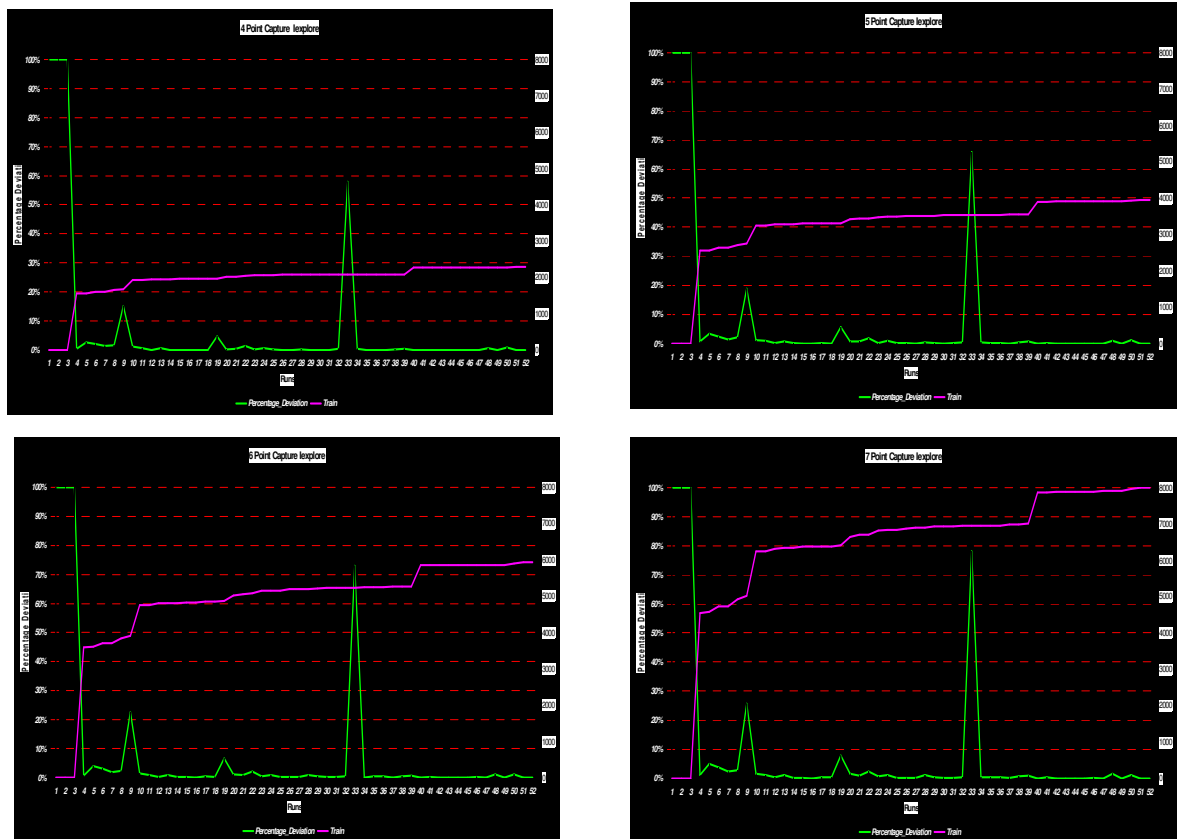



Figure 31: Chart – Deviation window (4-7)

Chapter 8. Deployment, Operations, Maintenance

8.1 Deployment

BASM application requires following pre-requisites:

- Install Microsoft .NET Framework version 2.0 or higher
- Install Perl ver 5.8.8 – Build 820 (Source : ActivePerl.org)
- Copy  into `“C:\Program Files\Common Files\ODBC\Data Sources”` folder to setup data grid display of event log file.

BASM application executables are installed in `“C:\SecurityMonitor\bin”` directory as shown in screenshot below.

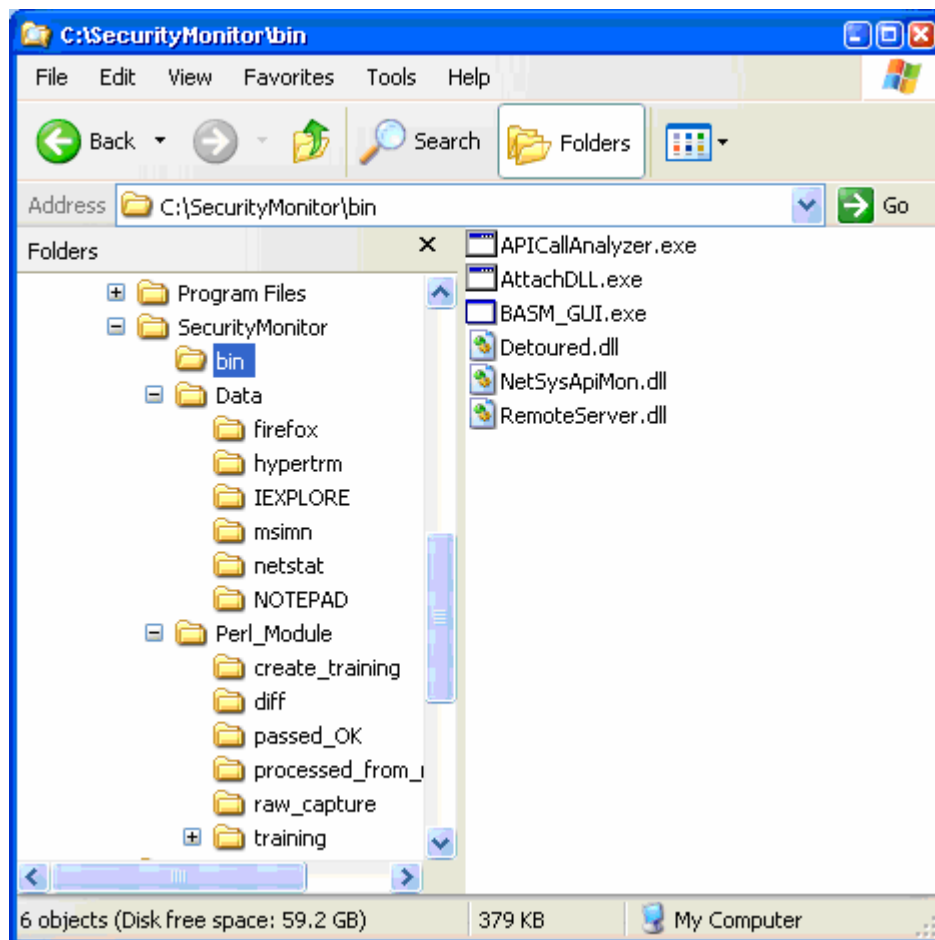


Figure 32: BASM.: Executables Source Code

BASM_GUI.exe starts the UI and it connects to APICallAnalyzer.exe Server(Starts APICallAnalyzer.exe server if it is not already running). AttachDLL.exe application attaches NetSysApiMon.dll with target application to intercept win32 system calls and their parameters. RemoteServer.dll implements the remote methods and APICallAnalyzer.exe creates remote object that allows GUI client to connect.

The “C:\SecurityMonitor\Perl_Module” directory contains the perl programs that do analysis using the API call log CSV files.

The “C:\SecurityMonitor\Data” directory contains all the API call log files used for analysis.

8.2 Operations

Start C:\SecurityMonitor\bin\BASM_GUI.exe to launch BASM GUI. Connect to APIMonAnalyzer Server using “Connection->Open Connection” menu option. BASM_GUI connects to Remote Server if it is already running, otherwise new instance will be launched. See following screen snapshot for details.

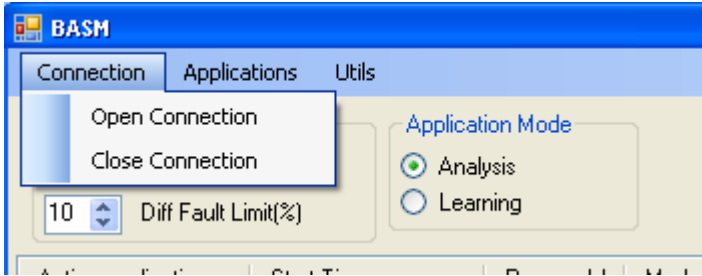


Figure 33: BASM : GUI connect/disconnect to server

When Remote GUI Client(BASM_GUI.exe) connects to server, its status is displayed as “Connected to Server”.

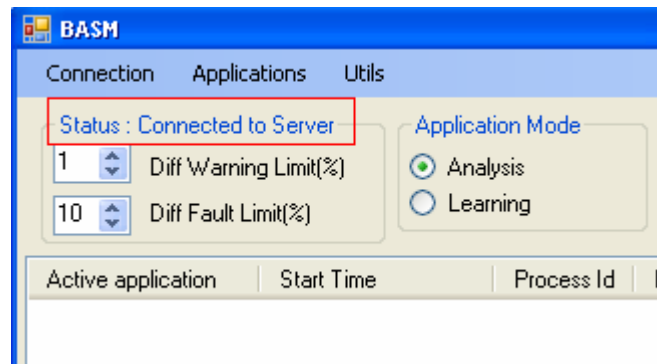


Figure 34: BASM Connection status

Application can be started to detour its system calls using “Applications->Start Application” menu option. Application can be started to collect data in learning mode or analysis mode. Learning mode collects data and stores in database to be used for analysis. In analysis mode when application ends its system call sequences are used for analysis to determine whether the application behavior is normal or not. Menu option “Applications->Manual Analysis” can be used to do manual analysis of any existing log file.

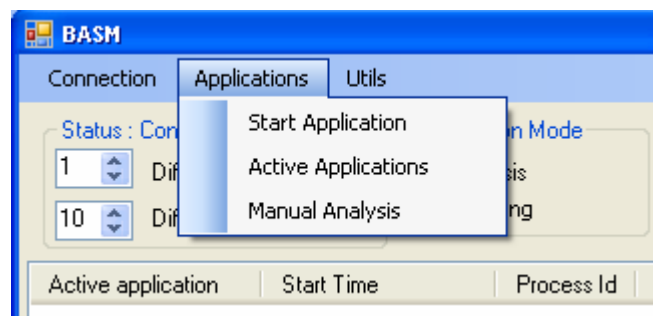


Figure 35: BASM : Application

Diff or deviation of system call sequences found by analysis module is reported as warning or fault event depending on user configured limits. This fault or warning event can be used to add diff or deviation sequences to learning database using “Add Diff to Learning Database” button. After diff or deviation data is added to learning database, next run with same sequence of API calls are treated as normal behavior.

The screenshot shows the BASH application window. At the top, there are tabs for 'Connection', 'Applications', and 'Utils'. Below the tabs, the status is 'Connected to Server'. There are two dropdown menus for 'Diff Warning Limit(%)' (set to 1) and 'Diff Fault Limit(%)' (set to 10). Under 'Application Mode', 'Analysis' is selected with a radio button, and 'Learning' is unselected. The file path is 'c:\SecurityMonitor\Data\EXPLORE\EXPLORE_EXE_2656.csv'. A table lists active applications with columns for 'Active application', 'Start Time', 'Process Id', and 'Mode'. Below this is an 'Event Log' section with a button labeled 'Add Diff to Learning Database'. At the bottom, a table displays event details with columns for 'Time', 'File', 'Status', 'Difference(%)', and 'Event Description'.

Time	File	Status	Difference(%)	Event Description
4/11/2007 8:33 ...	5_firefox_exe_2872.csv	Trace	0.07	firefox_exe_2872.csv : File passed Normally within the Warning Level
4/11/2007 8:33 ...	6_firefox_exe_2872.csv	Trace	0.11	firefox_exe_2872.csv : File passed Normally within the Warning Level
4/11/2007 8:33 ...	7_firefox_exe_2872.csv	Trace	0.14	firefox_exe_2872.csv : File passed Normally within the Warning Level
5/11/2007 1:17 ...	4_IEXPLORE_EXE_332.csv	Fault	40.45	IEXPLORE_EXE_332.csv : File requires Immediate attention
5/11/2007 1:17 ...	5_IEXPLORE_EXE_332.csv	Fault	42.63	IEXPLORE_EXE_332.csv : File requires Immediate attention
5/11/2007 1:17 ...	6_IEXPLORE_EXE_332.csv	Fault	44.34	IEXPLORE_EXE_332.csv : File requires Immediate attention
5/11/2007 1:17 ...	7_IEXPLORE_EXE_332.csv	Fault	45.76	IEXPLORE_EXE_332.csv : File requires Immediate attention
5/11/2007 1:18 ...	4_IEXPLORE_EXE_4980.csv	Fault	38.79	IEXPLORE_EXE_4980.csv : File requires Immediate attention
5/11/2007 1:18 ...	5_IEXPLORE_EXE_4980.csv	Fault	41.36	IEXPLORE_EXE_4980.csv : File requires Immediate attention
5/11/2007 1:18 ...	6_IEXPLORE_EXE_4980.csv	Fault	43.34	IEXPLORE_EXE_4980.csv : File requires Immediate attention
5/11/2007 1:18 ...	7_IEXPLORE_EXE_4980.csv	Fault	44.87	IEXPLORE_EXE_4980.csv : File requires Immediate attention

Figure 38: BASH : Adding Percentage Deviation to Normal Behavior

Chapter 9. Summary, Conclusions, and Recommendations

9.1 Summary

The project aims to capture application behavior and define it accurately. In hunt of this we initially had planned to include the network packet capturing along with system call tracing and argument analysis. But at the mid stage, we had to omit it citing the goal and revised project scope which aimed for more accuracy and more applications rather than a single application. The very first design only had window size of six where in we created a pattern of system calls in group of six. Also this implementation did not take threads and their scheduling into account. Although, not very satisfactory but this approach led us to the final destination. We then enhanced the content analyzer and set the window size floating from four to twenty five. Also we included thread id and started creating patterns based on the thread ids giving us desired and accurate results.

The project began with several milestones which were captured from time to time but the final milestone conquered was only set after achieving several other milestones. The insight to reach there was given by the incremental approach undertaken in developing this application.

9.2 Conclusions

Following are the conclusions that can be drawn by the detailed application behavior analysis performed on Internet Explorer, Mozilla Firefox and Outlook Express:

- Every application has a unique signature with the sequence of system calls per thread for a particular operation.
- Application behavior can be determined to be normal or abnormal by using its learning DB (created by history of past runs)
- As window size increases, we see better accuracy compared to window size of 4, and helps to detect abnormal behavior.
- As capture window size increases, learning DB records count also increases.
- Comparing one application data with other application learning DB results $>50\%$ deviation. As the window size increases, deviation also increases.
- As the window size increases time for analysis (deviation calculation) also increases.
- From the 3 applications behavior analysis, we find that Window size of 15 is optimum for current analysis algorithm.
- If window size is small (say 4), learning DB stabilizes faster with fewer runs and accuracy of deviation calculation reduces.
- When Window size is larger (say 25), learning DB stabilization requires many runs and accuracy of deviation calculation is not accurate either.

- Performance of the analysis takes a long time to perform as the window size increases. From our analysis so far conducted in this type of environment, we can see that until the window size reaches close to 14 to 16, run takes less than a minute. As the size increases, the time to calculate increases.
- With Window Size of 15, accuracy of deviation calculation is acceptable, analysis performance is good and learning DB size is not big (compared to window size of 25)
- This Project can be further enhanced by using timestamp for analysis, add more win32 API calls detoured, analyze the contents of API call parameters, and improve the performance analysis module.

9.3 Recommendations for Further Research

BASM is just a stepping stone to the goal of behavior based application monitors. It conquers many challenges and over comes with its simple yet versatile design which can be easily enhanced by further researchers. One approach BASM could not explore is timings of the system call and their durations. It would be interesting if one can take these timings into calculations along with system call sequences.

First, BASM does not use timestamp during analysis, but timestamp is available for each API call in the collected CSV log files. Timestamp can be used for more accurate and less false positive analysis by eliminating unrelated sequences and also detect botnet attack. One of the approaches for detecting botnet attack is to use API call timestamp to compare the frequency of API call sequences of normal application run with abnormal application run.

Second, BASM detours networking, file system, Registry and most common win32 API calls. Framework is already available in NetSysApiMon.cpp file for adding detour functions for more win32 API system calls. By adding more system calls, application behavior can be determined to be normal or abnormal more accurately.

Third, BASM framework is available for application API name and parameters to be serialized and sent to APICallAnalyzer server. Analyzing API calls with their parameters is more interesting and can help to understand application behavior better. Picking right set of API calls and their parameters is challenging. If API's data is encrypted, it is not easy to

interpret data portion of the system calls. If we assume data portion of the system call is not encrypted, we will be able to analyze the contents of the system calls.

Forth, BASM application currently needs operator to start application and navigate to create learning database. It requires many runs and lot of effort is needed to create stable learning database for accurate determination of application behavior. BASM can be enhanced to automate launching of application and navigate to create exhaustive learning database before application runs in production mode.

Fifth, BASM analysis module window size is hard-coded to be between 4 and 25. GUI can be enhanced to accept min and max value for window size from user. Analysis module needs to use the window size value set by user during analysis.

Finally, BASM analysis of system calls is done after application terminates. Analysis module can be enhanced to do analysis at run-time. This requires analysis module to be fast and less CPU intensive.

Glossary

System Monitor: Security monitor is a standalone system which captures all the system level activities.

System Call: The system procedures necessary to perform certain privileged tasks at the operating systems level.

Log: Log is the mechanism of storing records of various activities for monitoring and future reference.

Detours: Detours is a Win32 binary function library calls at the run time, developed by Microsoft.

DLL: Dynamic Link Library

GUI: Graphical User Interface

References

1. Geer, David
http://www.computer.org/portal/cms_docs_computer/computer/homepage/0306/r3014.pdf.
Retrieved March 18, 2007, from
http://www.computer.org/portal/cms_docs_computer/computer/homepage/0306/r3014.pdf
2. P. Astithas, V. Pappas, B. Maglaris, "Detecting Intrusions by Monitoring System Processes" , in Proceedings of the 8th HPOVUA Plenary Workshop on Network and Systems Management, Berlin, Germany, June 2001.
3. Lin, Frank, Shi, Hang, & Wang, Xiangrong (August, 2006). Preventing Outgoing Spam by Monitoring User Interactions. *1st International Symposium on Pervasive Computing and Applications*.
4. Shah, Swar (August 10, 2006). Automatic Security Monitor. Retrieved March 18, 2007, from SJSU fclin Web site: http://www.cmpe.sjsu.edu/~fclin/doc/automatic_security_monitor.doc
5. GAO, Debin (2007). Gray-Box Anomaly Detection using System Call Monitoring. *Dissertation report to CMU*