Active Change Auditing and Integrity Verification
System using MD5 Hashing

GRADUATE PROJECT TECHNICAL REPORT

Submitted to the Faculty of
the Department of Computing and Mathematical Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Ramesh Krishnamoorthy
Fall 2005

Committee Members

Dr. Mario Garcia
Committee Chairperson

Dr. David Thomas
Committee Member

Dr. Ajay Katangur
Committee Member
Active Change Auditing and Integrity Verification System using MD5 Hashing

GRADUATE PROJECT TECHNICAL REPORT

Submitted to the Faculty of
the Department of Computing and Mathematical Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Ramesh Krishnamoorthy
Fall 2005

Committee Members

Dr. Mario Garcia  ______________________________
Committee Chairperson

Dr. David Thomas  ______________________________
Committee Member

Dr. Ajay Katangur  ______________________________
Committee Member
Change monitoring is the definitive way of detecting intrusion. This intrusion detection system monitors a repository of data for change. The repositories could be anything from files to file systems or hosts to network devices. Each repository is fingerprinted using MD5. To verify integrity, the repository is fingerprinted again and compared with the stored fingerprints. If both fingerprints do not match, unauthorized changes have occurred.
# TABLE OF CONTENTS

Abstract .................................................................................................................. ii

Table of Contents ................................................................................................ iii

List of Figures ........................................................................................................ v

List of Tables .......................................................................................................... vi

1. Background and Rationale .............................................................................. 1
   1.1 Change Auditing ..................................................................................... 1
   1.2 Security Constraints ............................................................................. 1
      1.2.1 Integrity ......................................................................................... 1
      1.2.2 Confidentiality ............................................................................. 2
      1.2.3 Availability .................................................................................. 2
   1.3 Hashing .................................................................................................... 2
      1.3.1 Application of Hashing ................................................................. 3
      1.3.2 Hashing Techniques ................................................................... 3
   1.4 Message Digest 5 (MD5) ........................................................................ 4
   1.5 ISO 17799 / BS 7799 ............................................................................ 7
   1.6 Intrusion Detection System ................................................................... 9
   1.7 IDS in a Switched Environment ............................................................ 12
   1.8 Integrity Assurance ............................................................................. 15
      1.8.1 Importance of Integrity Assurance ............................................. 15

2. MD5Trip – A Narrative .................................................................................. 16
   2.1 Fingerprinting using MD5 .................................................................. 16
   2.2 Integrity Verification ............................................................................ 16
2.3 Change Auditing…………………………………………………………17
2.4 User Interface……………………………………………………………17

3. Proposed System Design………………………………………………19
  3.1 Environment…………………………………………………………19
  3.2 Objectives……………………………………………………………19
  3.3 MD5Trip Design……………………………………………………21
     3.3.1 MD5Create.pl………………………………………………23
     3.3.2 MD5Compare.pl…………………………………………….23

4. Testing and Evaluation…………………………………………………26
  4.1 Intrusion Detection…………………………………………………26
  4.2 Testing………………………………………………………………26
     4.2.1 Types of Testing……………………………………………26
  4.3 Benchmarks…………………………………………………………29

5. Future work and conclusion……………………………………………31

Bibliography and References………………………………………………33

Appendix A. MD5Trip Reports…………………………………………..40
LIST OF FIGURES

Figure 1.1 One MD5 Operation ............................................................5
Figure 1.2 BS7799 Process Chart..........................................................7
Figure 1.3 Risk Association Chart.........................................................8
Figure 1.4 IDS in a Switched Network..................................................13
Figure 1.5 IDS in a Switched Network..................................................13
Figure 1.6 IDS in a Switched Network..................................................14
Figure 1.7 IDS in a Switched Network..................................................14
Figure 2.1 File Based Intrusion Detection System.................................16
Figure 2.2 MD5Conf.pl – MD5Trip Configuration File..........................19
Figure 3.1 Security Product Development Cycle..................................18
Figure 3.2 Flowchart of MD5Trip Operation.........................................23
Figure 3.3 HTML Output of file integrity verifier.................................26
Figure 4.1 Denial of Service testing of MD5Trip.................................28
Figure 4.2 Functionality testing of MD5Trip.........................................28
Figure 4.3 Testing Activity at Operation and Maintenance Stages...........29
Figure 4.4 Sample of MD5Trip Report.................................................30
Figure 4.5 Size Vs. Time Graph............................................................30
LIST OF TABLES

Table 3.2 Status Decision ................................................................. 25
1. BACKGROUND AND RATIONALE

Security has been a major concern in the recent times. Until recently, much of the research attention is towards perimeter security. Perimeter security tools like firewalls and routers are designed to protect from unauthorized entry. While such tools protect from outsider attacks, they don’t protect from insider attacks, maliciousness or accident. It is almost impossible to detect such errors and problems without a change auditing/integrity verification system.

1.1 Change Auditing

Change Auditing is a documented procedure to monitor change. A snapshot of the known “safe” state is compared with the snapshot of the current state in order to detect changes. The frequency and “depth” of the snapshots can provide details and help to describe effects of the change. It can be as simple as a fingerprint hash that can detect an occurrence of a change, to complete imaging solutions that detect the actual change.

1.2 Security Constraints

Any security system is designed to protect integrity, availability, and confidentiality. They are collectively called the CIA triad or CIA constraints. Security attacks can be classified into one or more types of compromised CIA constraints.

1.2.1 Integrity

Integrity refers to the trustworthiness of the data or resources, and it is usually phrased in terms of preventing improper or unauthorized change. Integrity is often
discussed in two facets, data integrity and system integrity. Data integrity is preserved if
the data has not been altered or modified in an unauthorized manner while in storage,
during processing or while in transit. System integrity is described as the quality of a
system to perform the intended function in an unimpaired manner, free from
unauthorized manipulations [Bishop 2003].

1.2.2 Confidentiality

Confidentiality is the concealment of information or resources. The need for
keeping information secret arises from the use of computers in sensitive fields such as
government and industry. Confidentiality is often implemented by cryptographic
encryption [Bishop 2003].

1.2.3 Availability

Availability refers to the ability to use the information or resource desired.
Availability is an important aspect of reliability as well as of the system design because
an unavailable system is at least as bad as no system at all. Availability is most
vulnerable target of the three constraints, and crackers have exploited this with denial of
service attacks. [Bishop 2003].

1.3 Hashing

Integrity constraint poses a question “How to verify if the data or the system is what
it claims to be?” One method is to use one way hashing function. Hash function or a hash
algorithm is a function that takes an input data and generates an output hash or
fingerprint. The process of computing such a value is called hashing. One of the
important properties that hashing has is that two different inputs are unlikely to produce
the same output hash value. A fundamental property of all hash functions is that if two hashes (using the same hashing function) are different, then the two inputs are different in some way [Bishop 2003].

Typical hash functions have an infinite domain, such as byte strings of arbitrary length, and a finite range, such as bit sequences of some fixed length. Functions that follow this paradigm and produce random-looking output are considered stock hash functions, because they can be used directly in or easily adapted for most applications. Functions intended for cryptographic hashing, such as MD5, are commonly used as stock hash functions.

Hash functions that are one-to-one are also called randomization functions. Randomization functions cannot directly use stock hash functions, but instead use a series of reversible "mixing" operations on the function input to satisfy the required one-to-one property [Wikipedia 2005].

1.3.1 Application of Hashing

The ability of hashing algorithms to fingerprint a document can be used to uniquely identify the document. Any changes to the document will result in an entirely new hash key. In using this concept in reverse, it can be concluded that a new hash key indicates that the document is altered or modified.

1.3.2 Hashing Techniques

There are many one way hashing algorithm techniques. The most popularly used cryptographic hash functions are SHA1 and MD5 algorithms. These techniques take input in “fixed” sized blocks and compute a “fixed” size fingerprint of that input known as hash key. This hash key is always unique to the input data [Bishop 2003].
1.4 Message Digest Version 5 (MD5)

MD5 is an algorithm that is used to verify data integrity through the creation of a 128-bit message digest from data input, which may be a message of any length. Message digest produced using MD5 is claimed to be as unique to that specific data much as a fingerprint is to the specific individual. MD5, which was developed by Professor Ronald L. Rivest of MIT, is intended for use with digital signature applications [Rivest 1992]. MD5 is currently a standard, Internet Engineering Task Force (IETF) Request for Comments (RFC) 1321. According to RFC 1321, it is "computationally infeasible" that any two messages that have been input to the MD5 algorithm could have as the output the same message digest, or that a false message could be created through interception or sniffing of the message digest. MD5 is the third message digest algorithm created by Rivest [Whatis 2005].

MD5 processes a variable length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks; the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64-bit integer representing the length of the original message.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted $A$, $B$, $C$ and $D$. These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed $rounds$; each round is composed of 16 similar operations based on a non-linear function $F$,
modular addition, and left rotation. Figure 1 illustrates one operation within a round.

There are four possible functions; a different one is used in each round:

\[
F(X, Y, Z) = (X \land Y) \lor (\neg X \land Z) - 1
\]

\[
G(X, Y, Z) = (X \land Z) \lor (Y \land \neg Z) - 2
\]

\[
H(X, Y, Z) = X \oplus Y \oplus Z - 3
\]

\[
I(X, Y, Z) = Y \oplus (X \lor \neg Z) - 4
\]

\(\oplus, \land, \lor, \neg\) denote the XOR, AND, OR and NOT operations respectively.

![Figure 1.1 One MD5 operation [WikipediaMD5 2005]](image)

MD5 consists of 64 of these operations, grouped in four rounds of 16 operations.

\(F\) is a nonlinear function; one function is used in each round. \(M_i\) denotes a 32-bit block of the message input, and \(K_i\) denotes a 32-bit constant, different for each operation [WikipediaMD5 2005].
Using the MD5 hashing it is possible to verify state of integrity constraints. Traditionally, cryptographic hashing algorithms have been used for storing a user’s password in an encrypted form. The user can enter a variable length password, but that is not a desirable format for the storage of authentication data records. A hash of the password that the user enters is generated by a convenient hashing algorithm and that encrypted text is stored in the authentication file. The hash key generated can be produced again only by using the password as the input. For example, MD5 hashing algorithm computes the message digest, for the word “password” as “5f4dcc3b5aa765d61d8327deb882cf99”. This message digest is unique and can be computed if and only “password” is the input for the MD5 algorithm[RFC1321].

Data integrity in a database can be verified using hashing algorithms. A hash for every row and column, across the table are generated. A hash of each database is generated. When data corruption or alteration occurs, the database hashes are recomputed and the corrupt database is identified. Every row and column is used as input, and hashes are recalculated. Elements with error cause the hashes for their corresponding rows and columns to change. For example if the row hash of the fifth row and the column hash of the third row has changed then it can concluded that the item at the fifth row and the third column is not what it used to be.

Change auditing software helps a user to build a foundation of security by providing a way to quickly detect accidental or malicious changes to data and recover to a desired safe good state. Change auditing software, when deployed with other regular network security tools, can detect and provide advantages such as snapshot comparisons,
lower costs of operation, maximize uptime and increase stability and control over a known good state.

1.5 ISO 17799 / BS 7799

In the wake of the problems encountered in security domains, standards were set out for requirements for an information security management system. This document helped to identify, manage, and minimize threats to information. The goal of BS7799 / ISO 17799 is to “provide a common base for developing organizational security standards and effective security management practice and to provide confidence in inter-organizational dealings.” Figure 1.2 explains the steps involved in the process of getting certified as BS7799/ISO17799 compliant.

![Figure 1.2 BS7799 Process Chart [BS7799 2005]](image)

For any organization that stores confidential information on internal or external systems, depends on such systems to run its operations, a scale to prove their ability to protect such information would be of great interest. BS7799 / ISO 17799 are a set of standards that assures, a “certified company” has the ability and resources to protect
information assets as per the security levels described in the standard. Figure 1.3 represents the risk associated with some industries.

![Figure 1.3 Risk Association Chart [BS7799 2005]](image)

Obviously, complying with the ISO 17799 standard or obtaining BS7799-2 certification does not prove that an organization is 100% secure. The truth is, barring a cessation of all activity, there is no such thing as complete security. Nevertheless, adopting this international standard confers certain advantages that any manager should take into consideration, at the organizational level, certification serves as a guarantee of the effectiveness of the effort put into rendering the organization secure at all levels, and demonstrates the due diligence of its administrators. Compliance, at a legal level is achieved as a certification demonstrates to competent authorities that the organization observes all applicable laws and regulations. In this matter, ISO1779/BS7799 complements other existing standards and legislation (for example HIPAA, the Privacy Act of 1974, the Computer Security Act of 1987, the National Infrastructure Act of 1996, the Gramm-Leach-Bliley Act of 1999, and the Government Information Security Reform Act of 2001). Risk management leads to a better knowledge of information systems, their weaknesses and how to protect them, at the operating level. Equally, it ensures a more dependable availability of both hardware and data. At the commercial level, credibility
and confidence of the partners, shareholders and customers are reassured when they see the importance afforded by the organization to protecting information. Certification can help set a company apart from its competitors and in the marketplace. Already, international invitations to tender are starting to require ISO 17799 compliance. At the financial level, reduced costs related to security breaches, and possible reduction in insurance premiums [BS7799 2005].

1.6 Intrusion Detection Systems

Intrusion detection is needed in today’s computing environment because it is very difficult to keep pace with the current and potential threats and vulnerabilities in our computing systems. The environment is constantly evolving and changing fueled by new technology and the Internet. To make matters worse, threats and vulnerabilities in this environment are also constantly evolving. Intrusion detection products are tools to assist in managing threats and vulnerabilities in this changing environment.

Threats are people or groups who have the potential to compromise computer systems. These may be a curious teenager, a disgruntled employee, or espionage from a rival company or a foreign government. The hacker has become a nemesis to many companies.

Vulnerabilities are weaknesses in the systems. Vulnerabilities can be exploited and used to compromise your computer systems. New vulnerabilities are discovered everyday. Every new technology, product, or system brings with it a new generation of bugs and unintended conflicts or flaws. Also the possible impacts from exploiting these vulnerabilities are constantly evolving. In a worst-case scenario, an intrusion may cause
production downtime, sabotage of critical information, theft of confidential information, 
cash, or other assets, or even negative public relations that may affect a company’s stock 
price.

Intrusion detection products are tools that can assist in protecting a company from 
intrusion by expanding the options available to manage the risk from threats and 
vulnerabilities. Intrusion detection capabilities can help a company secure its information. 
The tool could be used to detect an intruder, identify and stop the intruder, support 
investigations to find out how the intruder got in, and stop the exploit from use by future 
intruders. The correction should be applied across the enterprise to all similar platforms. 
Intrusion detection products can become a very powerful tool in the information security 
practitioner’s tool kit.

ID stands for Intrusion Detection, which is the art of detecting inappropriate, 
incorrect, or anomalous activity. ID systems that operate on a host to detect malicious 
activity on that host are called host-based ID systems, and ID systems that operate on 
network data flows are called network-based ID systems.

Sometimes, a distinction is made between misuse and intrusion detection. The 
term intrusion is used to describe attacks from the outside; whereas, misuse is used to 
describe an attack that originates from the internal network. However, most people don't 
draw such distinctions. The most common approaches to ID are statistical anomaly 
detection and pattern-matching detection [Lehmann 2005].

It is very difficult to compromise a system without altering a system file, so file 
integrity checkers are an important capability in intrusion detection. A file integrity 
checker computes a checksum for every guarded file and stores this. A checksum can be
generated at a later time and can be tested against the current value against the stored value to determine if the file has been modified. A file integrity checker is a capability that one should expect to receive with any commercial host based intrusion detection system.

The primary checksum that was used in this research was a 32 bit CRC (Cyclic Redundancy Check). Attackers have demonstrated the ability to modify a file in ways, such as adding multiple white spaces, the CRC checksum could not detect the changes, so stronger checksums known as cryptographic hashes are recommended. Examples of cryptographic hashes include MD5 and SHA1.

When updating files or applying system patches files are update or changed, although this is a desired change. The file integrity checker can not classify it as a desired change and generates an error. Creating the initial database of signatures is easy; keeping it up to date is much harder. However, even if the integrity checker was run only once, after the first install, the system this can still be very valuable. If there is ever concern that the system was compromised, the checker can be run again to determine which files have, or have not been modified.

The other challenge with a file integrity checker is that a pristine system should be used when the first reference database is created. Otherwise one may be creating cryptographic hashes of a compromised system while feeling warm and fuzzy for implementing good security. It is also very important that the reference database is stored offline or an attacker may be able to compromise the system and hide their tracks by modifying the reference database [SANS 2005].
1.7 IDS in a Switched Environment

The difficulty of implementing IDS in a switched environment stems from the basic differences between standard hubs and switches. Hubs have no concept of a connection and thus will echo every packet to every port on the hub, excluding only the port the packet used to enter the network. A switch operates differently by making a virtual connection between the two hosts and forwards the packet to that one host. In a hub environment our sensors can be placed almost anywhere, while with switches specific hardware workarounds must be used to assure the sensor is able to see the traffic required. The current options are taps, hubs and spanning ports.

A spanning port configures the switch to behave like a hub for a specific port. For instance in Figure 1.4, the connection between the switch and the resource machine is to be monitored. To do this switch must be made to span or send the data from the resource machines port to the IDS port. This can be done with transmit data, receive data or both. Some current switches cannot be relied on to pass 100% of the traffic to the spanned port, so attacks could go un-noticed even when the IDS system is configured to look for the attack. Switches only allow one port to be spanned at a time, so monitoring multiple machines can be difficult.

![Figure 1.4 IDS in a Switched Network](Laing 2005)
Using hubs or taps is a very similar solution. The hub or tap is placed between the connections to be monitored. This is usually between two switches, a router and switch, or a server and switch, etc. In Figure 1.5 a hub has been placed between the resource machine and the switch. This allows traffic to still flow between the switch and the resource while the properties of the hub cause a copy of the traffic to be copied off to the IDS. This scenario, like the span port is only suitable for single machines. Multiple machines on the hub would cause network problems and remove the benefits of a switched solution. In addition, to get a fault tolerant hub would increase the cost of the solution dramatically. Taps are by design fault tolerant having the main connection (i.e. the connection between the resource and the switch), hardwired into the device, preventing failure due to loss of connectivity.

Figure 1.5 IDS in a Switched Network [Laing 2005]

Figure 1.6 shows a tap monitoring a single resource machine. The tap is unidirectional and passes traffic from the switch and resource machine to the IDS only.
This does not prevent traffic passing from the IDS to the switch or resource machine nor can traffic be directed at the IDS. Since the tap is unidirectional, the traffic can route from several taps back to a hub to be monitored by the IDS system, without causing network problems, this is shown in Figure 1.7.

The above section “IDS in Switched Environment” has been adapted from SANS FAQs [SANS 2005].

1.8 Integrity Assurance

Integrity assurance can be a key element of business’s risk management. Integrity management solutions for IT security and operations staff can reduce operational risks by
effectively controlling changes to IT systems, regardless of its source and location. Managing integrity is essential to ensure systems security, audit and compliance and in “closing the loop” on change and configuration management processes. Integrity management software reduces operational risk and assures the integrity of systems enterprise-wide by ensuring the security of systems, instilling accountability for change, providing visibility across the enterprise and increasing the availability of information services [Tripwire 2005].

1.8.1 Importance of Integrity Assurance

There have been a lot of legal changes, recently, in the areas of information security and computing, legislation for handling and protection of medical information, and handling and protection of financial data. For example, laws such as the Health Insurance Portability and Accountability Act (HIPAA) have sections that address the privacy liabilities, of the patient’s data stored electronically, to the health care organizations [HIPAA 2005]. With such laws in practice, it is not only important for an organization to protect the data; they also become liable for its misuse [California 2005].
2. MD5TRIP – NARRATIVE

This project is designed to monitor integrity constraints. MD5Trip is a file level change auditing and integrity verification tool. When an attacker does manage to breach into the entire levels of security, it is extremely difficult to stop an attack. The last line of defense is to use a file level integrity monitor that helps to identify the details of the attack. The audit logs can help a system administrator as a detective control, to identify file changes and take corrective action to prevent a similar attack from happening again. Figure 2.1 describes integrity monitor and change auditing engine working together as an Intrusion Detection System (IDS).

![Diagram of Intrusion Detection System](image)

**Fig 2.1 File Based Intrusion Detection System**

2.1 Fingerprinting using MD5

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the
difficulty of computing with two messages having the same message digest is on the order of $2^{64}$ operations, and that the difficulty of computing any message having a given message digest is on the order of $2^{128}$ operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is, of course, justified, as is the case with any new proposal of this sort [Rivest 1992].

### 2.2 Integrity Verification

Integrity verification is the method of proving that two instances of data are the same. In other words, Integrity verification is the method by which a user can confirm the data that is available, is exactly the same data the user wants. The author of the data fingerprints a document using the MD5 algorithm and stores the message digest in two different locations one considered safe and another public. Any time the integrity of the document needs to be verified the document is hashed again using MD5. The message digests are compared. If the message digests match, the file is exactly what the author created. Otherwise, the document is considered to be altered.

In the proposed program the user will have an option to select from running the MD5 algorithm on the file and matching the message digest to available message digest or to actively monitor integrity on a selected folder.

When monitoring integrity for a selected folder the program automatically generates the MD5 message digests for each file in the folder and saves the digests in a secure location. This process is repeated at various intervals of time and activity. This ensures any change is noticed and described to the systems administrator.
2.3 Change Auditing

All the activities that are performed by MD5Trip are audited to an append-only log file, by the change auditing module. The change auditing engine is a deterrent and detective control implemented to discourage the attacker and to account for the actions of the attacker. The log files also provide a method for analyzing performance metrics of the program.

2.4 User Interface

A user interface is a set of commands or menus through which a user communicates with a computer system. A command-driven interface is one in which user enters commands. A menu-driven interface is one in which the user select command choices from various menus displayed on the screen. The user interface is one of the most critical parts of any system because it determines how easily a user can make the program do what was intended to do [Martin 1988]. The project is developed to be an administrator’s tool and has the ability to be run from a command prompt. The configurations for the program are stored in a file called MD5Conf.pl. The user needs to define the path of the monitored folder and the path where reports should be generated. Then the user can double click on the create.pl and compare.pl files to execute them. A screenshot of the configuration file is in figure 2.2.
#!/usr/bin/perl -w
package md5conf;

use strict;

use vars qw($dbpath);
use vars qw($path);
use vars qw($report_file);

$dbpath = "f:/project/dbfile.db"; #Database File
$path = "C:/windows/system32"; #Integrity Path
$report_file="f:/project/reports/md5Trip_"; # Report filename

Figure 2.2 MD5Conf.pl- MD5Trip's Configuration file
3. SYSTEM DESIGN

3.1 Environment

For this project, Windows XP running Active State Perl version 5.8.7.813 is used. A default installation of Windows XP and Perl is sufficient for core functionality. Since MS-DOS Console does not support ANSI based output an ANSI to MS-DOS conversion is required. ANSI-color module from CPAN is included. The complete code and operation of this project can be ported to any operating system as long as a Perl interpreter is available for it. Perl interpreters are available for most popular operating systems.

3.2 Objectives

The objective of this project is to be able to compare and verify the integrity constraints at any two intervals of time. A test folder will act as a smaller model of a complete “operational system” for this project. The same method can be extended to any size of data, with minimal modifications to the code. The folder is used to minimize time per run constraint. This folder is where the integrity constraints are needed to be verified. A Web server is an example of such a setup, where all files exist under one folder. It is impossible for a Web administrator to go through thousands of Web pages trying to identify unauthorized changes. This project can be used to identify such changes very efficiently. This project can help to identify files which are altered since the last fingerprint.
Typically, any security related project is developed using the secure system development life cycle as described in Figure 3.1. In the system initiation stage the specifications, purpose, mission and configuration of the system that will be built are defined.

![Security Product Development Life Cycle](image)

**Figure 3.1 Security Product Development Life [NIST 2003]**

In the development and acquisition stage the system is possibly contracted and constructed according to documented procedures and requirements. In the implementation and installation stage, the system is installed and integrated with other applications, usually on a testing environment, similar to a production environment. Operational and maintenance phase is where the system is operated and maintained according to its mission requirements. Disposal is the last phase of security product’s lifecycle, the product is completely deactivated and removed from the computer, network
and everywhere else it exists. Then tests are performed to make sure that no residual files or program integration can create any further vulnerabilities.

A fingerprinting message digest is computed for each file in the test folder and stored in a safe location. It is assumed the message digest data is protected by not being available for public access. Every time a Web administrator changes any pages, this program should be run to compute new hashes. This program can then use these message hashes to identify any unauthorized changes.

### 3.3 MD5Trip Design

This project contains two software modules. The first module *MD5Create* computes the MD5 hashes of the files in the folder and should be run after every authorized change made. The second module *MD5Compare* generates a report of the status of integrity constraints. This module should be run every time the user needs to verify the integrity of the test folder. Figure 3.1 describes the process of file integrity checking. The two modules work in parallel to fingerprint and check changes.
Figure 3.2 Flowchart of MD5Trip operation
3.3.1 MD5Create.pl

MD5Create module is used to compute message digest for all the files in the test folder. The computed hash values are securely stored in a database outside the test folder. This database is created to hold the name of the file and its associated 128 bit message digest. This database is a snapshot of the monitored folder; therefore care must be taken to ensure the system is free from virus or other malicious code at least till the first run. This will act as a reference for a pristine and desired state of the test folder. This database needs to be repopulated with new message digests every time a known authorized change occurs, by running this module. This ensures the changes are recorded in the database and the comparisons do not result in false positives.

3.3.1 MD5Compare.pl

MD5Compare module is the primary integrity verification and auditing module. This module can operate independently of the MD5Create module as long as the database is accessible. This module computes message digests for each file in the monitored folder. The newly generated message digests are compared that with the existing message digest values stored in the database. The newly generated message digest and existing message digests can match or differ. A match in the message digests indicate that the file is no different from the last time it was fingerprinted using MD5 message digest algorithm. Although one cannot completely rule out the possibility that no changes have been made and undone after the database creation.

If the message digests differ, it is an indication that the files have been altered or modified since the database was created. If files have been deleted, a dangling message
digest from the database would occur with no matching filename and message digest in the current file set. If new files have been created, the reverse would occur with dangling message digest in current file set without a match for it in the database. The scenarios and their interpretations are summarized in table 3.1

Table 3.1 Status Decision

| State      | Comments   | Status | H1=Message Digest from the database  
| H2=Message Digest from the current run  
| Φ=No Message Digest |
|------------|------------|--------|--------------------------------------------------------------------------------|
| H1=H2      | No Change  | OK     |                                                                               |
| H1!=H2     | Altered    | Failed |                                                                               |
| H1=Φ;H2    | New File   | Failed |                                                                               |
| H1;H2= Φ  | File Deleted | Failed |                                                                               |

A HTML based report is generated based on table 3.1 for testing and analysis of the product and the computer’s integrity.
Altered Files

C:\windows\system32\config\AppEvent.EVT
C:\windows\system32\spnmsg.dll
C:\windows\system32\CatRoot2\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\TimeStamp
C:\windows\system32\wbem\RepositoryFS\MAPPING2.MAP
C:\windows\system32\CatRoot2\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\catdb
C:\windows\system32\config\SysEvent.EVT
C:\windows\system32\config\systemprofile\ntuser.dat
C:\windows\system32\inetsrv\MetaBase.bin
C:\windows\system32\wbem\Logs\wmiprov.log
C:\windows\system32\wbem\RepositoryFS\INDEX.BTR
C:\windows\system32\config\SecEvent.EVT
C:\windows\system32\wbem\RepositoryFS\OBJECTS.MAP
C:\windows\system32\CatRoot2\edb.chk
C:\windows\system32\wbem\RepositoryFS\MAPPING1.MAP
C:\windows\system32\CatRoot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\TimeStamp
C:\windows\system32\config\systemprofile\ntuser.dat.LOG
C:\windows\system32\CatRoot2\dberr.txt
C:\windows\system32\wbem\RepositoryFS\OBJECTS.DATA
C:\windows\system32\wbem\RepositoryFS\INDEX.MAP
C:\windows\system32\wbem\Logs\Framework.log
C:\windows\system32\wbem\Logs\wbemness.log

New Files

C:\windows\system32\LegitCheckControl.dll
C:\windows\system32\framesh.txt
C:\windows\system32\CatRoot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\WGA.cat

Missing Files

C:\windows\system32\CatRoot\edb.log
C:\windows\system32\LegitCheckControl.DLL

Process ended at Fri Mar 24 11:40:04 2006

Total Runtime: 72 Seconds

Figure 3.3 HTML Output of file integrity verifier
4. TESTING AND EVALUATION

4.1 Intrusion Detection

This project is tested to detect changes, in files, folders and filenames. When the MD5Create module is run, it generates a report containing a list of all files and their status, changed or not changed. For testing this project a production system with a routable IP address is used and “C:\windows\system32” system folder is monitored.

4.2 Testing

The file integrity checker can be put to test using various penetration tests. A test entirely depends on the scope of operation i.e. the level of the intrusion is directly related to the scope. For example, it would be enough to identify the vulnerability, in case of a non productive system or in situations where the cost of a safeguard would be more than the benefit produced by the system. Therefore, a security professional should choose the right type of test based on the scope of the vulnerability.

4.2.1 Types of Testing

Denial of service testing involves attempting to exploit specific weakness by bombarding that weakness till system resources are exhausted and the system performance starts to deteriorate or even stop responding, to all requests. The performance of MD5Trip is studied under extremely limited system resources. MD5Trip is tested in a production environment. The program uses less than 6MB of physical memory during its regular runs. In Figure 4.1, the process is running topping out at about 10MB using all the resources it needed. In the background, the program is
running intensively computing message digests for each file. An advantage of Perl as the
choice of programming is memory management is handled internally. Perl makes

![Windows Task Manager](image)

**Figure 4.1 Denial of Service testing of MD5Trip**

...sure the program does not hog the resources and cause a denial of service.

Functionality testing involves testing the systems responsible for applications
functionality as presented to a user. This will require testing the user’s input. The input is
validated for bad options, buffer overflows, etc. that can produce unexpected results. The
transaction of data between the processes, ensuring that the application performs to
specification and doesn’t permit the user to abuse the program to generate unexpected
results is tested by providing random inputs to the program. The program is designed to accept input only from a configuration file. The use of configuration file makes sure the user can just execute the program. As in Figure 4.2, the tests did not result in any unusual or unexpected responses; instead the program did not begin execution and displayed an error message.

![Figure 4.2 Functionality testing of MD5Trip](image)

File type testing involves testing the program to fingerprint different types of files, without any glitches such as exceptions, buffer under runs, etc. The low level file system details are beyond the scope of the program. The user is expected to be security trained, not to “force open” files that do not belong to that operating system.

As in Figure 4.3, scheduled testing and evaluation (ST&E) involves testing the product to under regular intervals of time to make sure the performance and functionality
have not changed.

**Figure 4.3 Testing Activity at the operation and maintenance stages [NIST 2003]**

If it has changed, maintenance is performed and bugs are traced and then corrected. This testing has been included just to retain the completeness of the testing stages. The program is tested at various instances of time, the program responded the same way every time.

Disposal testing, finally when the product is either considered obsolete or not useful to its mission, the product is “uninstalled” and general referencing tests and memory residual tests are performed to make sure no side effects occur to the functionality or performance of the system [NIST 2003]. The program uses Perl interpreter and the termination of the program is absolute as the interpreter terminates. The details of the Perl interpreter are beyond the scope of the paper.

Finally, the program is tested to make sure it could identify the files that have changed since the last fingerprint. With firewalls and network based security devices in places, it is extremely difficult for a hacker to alter the file system. Given the time restriction, waiting for a real hacker to attack is not possible. Hence, the various changes created by the operating system in the system folder are studied. The reason for selecting
the system32 folder is because it is the most important folder in a Windows based machine. This folder contains “dll”, “exe”, and the other configurations files for proper operation of the operating system. This folder contains maximum number of files, making it a favorite place for a hacker to hide rouge programs. Disrupting one file can result in total failure, like a popular hacker’s motive.

A baseline message digest database was computed for “c:\windows\system32”. This database contained the message digests for all the files in that folder. The folder was monitored at regularly on daily intervals to audit the changes that occur in Windows XP as regular operations. The changes to the files are limited to log files which change ever so frequently.

There are patches released by Microsoft whenever vulnerabilities are known. These patches are updates to various files in the system32 folder. The program revealed the changes to the file system created by applying these patches. In Figure 4.4, a list of the files that are changed after a patch was applied as revealed by MD5Trip is shown. The changes are categorized into altered, missing or new files. With an application of a patch about 25 files where altered. After the last trial, about 100 files had been altered since the first time the folder was fingerprinted. Complete reports are listed in appendix A.
4.3 Benchmarks

The performance metrics of this project is evaluated in terms of time. Time taken to fingerprint using MD5, time taken to verify the fingerprint, and the time taken to generate the report all the errors count towards the time that is mapped against the data size. The benchmarking will be performed for various data sizes, incrementing in Megabytes. The graphs will be plotted to display the results graphically. The results from running the program to fingerprint various sizes of data are seen in the Figure 4.5.
Figure 4.5 Size Vs. Time Graph
5. CONCLUSIONS

MD5Trip is an effort to audit changes and verify integrity. This is just a small step in implementing enterprise level integrity assurance; such an assurance can be achieved on by defense in depth. Limitations to integrate and package this tool into a administrator “Swiss-knife” toolkit would extremely useful. The program is designed to work with one specific file or directory path. It would be efficient to include selected files into the fingerprinting database. Message digests for only needed files can be computed, this would reduce the time taken to complete the execution of the program. One of the most useful features would be to integrate a file level integrity verification system, such as MD5Trip, into the operating system to authenticate every file. This would allow any unauthorized changes to be easily detected.

Managing integrity is essential to ensure systems security, audit and compliance. This project delivers integrity monitoring solution for IT security and operations staffs so they can reduce operational risk by effectively controlling change to IT systems, regardless of the source. Integrity monitoring reduces operational risk and assures the integrity of systems enterprise-wide by ensuring the security of systems.

This project audited and created accountability for changes in the system. This project identified unauthorized changes, by comparing the selected folders for changes. The project is evaluated in terms of its ability to detect unauthorized changes. Auditing for access and state of the system can account for confidentiality and availability of the system.
APPENDIX A - MD5Trip Reports

Process started at Fri Mar 24 11:38:52 2006

Altered Files
C:/windows/system32/config/AppEvent.Evt
C:/windows/system32/spmsg.dll
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp
C:/windows/system32/wbem/Repository/FS/MAPPING2.MAP
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/catdb
C:/windows/system32/config/SysEvent.Evt
C:/windows/system32/config/systemprofile/ntuser.dat
C:/windows/system32/inetsrv/MetaBase.bin
C:/windows/system32/wbem/Logs/wmiprov.log
C:/windows/system32/wbem/Repository/FS/INDEX.BTR
C:/windows/system32/config/SecEvent.Evt
C:/windows/system32/wbem/Repository/FS/OBJECTS.MAP
C:/windows/system32/CatRoot2/edb.chk
C:/windows/system32/wbem/Repository/FS/MAPPING1.MAP
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp
C:/windows/system32/config/systemprofile/ntuser.dat.LOG
C:/windows/system32/CatRoot2/dberr.txt
C:/windows/system32/wbem/Repository/FS/OBJECTS.DATA
C:/windows/system32/wbem/Repository/FS/INDEX.MAP
C:/windows/system32/wbem/Logs/FrameWork.log
C:/windows/system32/wbem/Logs/wbemess.log

New Files
C:/windows/system32/LegitCheckControl.dll
C:/windows/system32/ramesh.txt
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/WGA.cat

Missing Files
C:/windows/system32/CatRoot2/edb.log
C:/windows/system32/LegitCheckControl.DLL

Process ended at Fri Mar 24 11:40:04 2006

Total Runtime: 72 Seconds.
Process started at Mon Mar 27 16:19:33 2006

Altered Files
C:/windows/system32/config/AppEvent.Evt
C:/windows/system32/spmsg.dll
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp
C:/windows/system32/wbem/Repository/FS/MAPPING2.MAP
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/catdb
C:/windows/system32/config/SysEvent.Evt
C:/windows/system32/wpa.db1
C:/windows/system32/config/systemprofile/ntuser.dat
C:/windows/system32/inetsrv/MetaBase.bin
C:/windows/system32/wbem/Logs/wmiprov.log
C:/windows/system32/CatRoot2/edb.log
C:/windows/system32/wbem/Logs/wbemprox.log
C:/windows/system32/wbem/Repository/FS/INDEX.BTR
C:/windows/system32/config/SecEvent.Evt
C:/windows/system32/wbem/Repository/FS/OBJECTS.MAP
C:/windows/system32/wbem/Repository/FS/MAPPING1.MAP
C:/windows/system32/CatRoot2/edb.chk
C:/windows/system32/CatRoot2/TimeStamp
C:/windows/system32/config/systemprofile/ntuser.dat.LOG
C:/windows/system32/CatRoot2/dberr.txt
C:/windows/system32/wbem/Repository/FS/OBJECTS.DATA
C:/windows/system32/wbem/Repository/FS/INDEX.MAP
C:/windows/system32/wbem/Logs/FrameWork.log
C:/windows/system32/wbem/Logs/wbemess.log

New Files
C:/windows/system32/LegitCheckControl.dll
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/WGA.cat

Missing Files
C:/windows/system32/LegitCheckControl.DLL


Total Runtime:115 Seconds.
Process started at Thu Mar 30 12:07:43 2006

Altered Files
C:/windows/system32/config/AppEvent.Evt
C:/windows/system32/wbem/Logs/FrameWork.lo_
C:/windows/system32/spmsg.dll
C:/windows/system32/ntkrnlpa.exe
C:/windows/system32/fltlib.dll
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp
C:/windows/system32/wbem/Repository/FS/MAPPING2.MAP
C:/windows/system32/config/SysEvent.Evt
C:/windows/system32/PerfStringBackup.INI
C:/windows/system32/wbem/Logs/mofcomp.log
C:/windows/system32/wpa.db
C:/windows/system32/config/systemprofile/ntuser.dat
C:/windows/system32/Inetsrv/MetaBase.bin
C:/windows/system32/wbem/Log/wsmpirov.log
C:/windows/system32/wbem/Logs/wbemprox.log
C:/windows/system32/wbem/Repository/FS/INDEX.BTR
C:/windows/system32/ntoskrl.exe
C:/windows/system32/config/SecEvent.Evt
C:/windows/system32/wbem/Repository/FS/OBJECTS.MAP
C:/windows/system32/CatRoot2/edb.chk
C:/windows/system32/wbem/Logs/wbemess.lo_
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/E6024EAC88E6B6165D49FE3C95ADD735
C:/windows/system32/wbem/Repository/FS/MAPPING1.MAP
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp
C:/windows/system32/dllcache/fltlib.dll
C:/windows/system32/wbem/Logs/wmiadap.log
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/E6024EAC88E6B6165D49FE3C95ADD735
C:/windows/system32/dllcache/fltmgr.sys
C:/windows/system32/config/systemprofile/ntuser.dat.LOG
C:/windows/system32/CatRoot2/dberr.txt
C:/windows/system32/wbem/Repository/FS/OBJECTS.DATA
C:/windows/system32/dllcache/fltmc.exe
C:/windows/system32/wbem/Repository/FS/INDEX.MAP
C:/windows/system32/wbem/Logs/FrameWork.log
C:/windows/system32/wbem/Logs/wbemess.log

New Files
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/60E31627FDA0A46932B0E5948949F2A5
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/60E31627FDA0A46932B0E5948949F2A5
C:/windows/system32/pxwma.dll
C:/windows/system32/dllcache/fltmc.exe
C:/windows/system32/wbem/AutoRecover/679BDCDA90EDF24EC83945D38F2EEEAA.mof
C:/windows/system32/DRVSTORE/mpfilter_4C2E187DB623934A79E43DF6CA5ABBD5B357
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/oem3.CAT
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/A8FABA189DB7D25FBA7CAC806625FD30
C:/windows/system32/ssleay32.dll
C:/windows/system32/drivers/pxhelp20.sys
C:/windows/system32/Logfiles/W3SVC1/ex060328.log

---

**Missing Files**

C:/windows/system32/drivers/fltMgr.sys
C:/windows/system32/fltMc.exe
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/catdb
C:/windows/system32/CatRoot2/{127D0A1D-4EF2-11D1-8608-00C04FC295EE}/catdb
C:/windows/system32/CatRoot2/edb.log
C:/windows/system32/LegitCheckControl.DLL

---

**Process ended at Thu Mar 30 12:12:06 2006**

---

Total Runtime: 263 Seconds.
**Process started at Mon Apr 3 10:23:59 2006**

---

**Altered Files**

- `C:/windows/system32/config/AppEvent.EVT`
- `C:/windows/system32/wbem/Logs/FrameWork.log`
- `C:/windows/system32/spmsg.dll`
- `C:/windows/system32/ntkrnlpa.exe`
- `C:/windows/system32/fltlib.dll`
- `C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp`
- `C:/windows/system32/wbem/Logs/mofcomp.log`
- `C:/windows/system32/CatRoot2/{127D0A1D-4EF2-11D1-8608-00C04FC295EE}/catdb`
- `C:/windows/system32/wpa.dbl`
- `C:/windows/system32/config/systemprofile/ntuser.dat`
- `C:/windows/system32/inetsrv/MetaBase.bin`
- `C:/windows/system32/wbem/Logs/wmiprov.log`
- `C:/windows/system32/CatRoot2/edb.log`
- `C:/windows/system32/wbem/Logs/wbemprox.log`
- `C:/windows/system32/wbem/Repository/FS/INDEX.BTR`
- `C:/windows/system32/ntoskrnl.exe`
- `C:/windows/system32/config/SecEvent.EVT`
- `C:/windows/system32/wbem/Repository/FS/OBJECTS.MAP`
- `C:/windows/system32/CatRoot2/edb.chk`
- `C:/windows/system32/wbem/Logs/wbemess.log`
- `C:/windows/system32/config/systemprofile/Application Data/Microsoft/CryptnetUrlCache/Content/60E31627FDA0E46932B0E5948949F2A5`
- `C:/windows/system32/dllcache/ntuser.dat.LOG`
- `C:/windows/system32/ntoskrnl.exe`
- `C:/windows/system32/CatRoot2/dberr.txt`
- `C:/windows/system32/wbem/Repository/FS/OBJECTS.DATA`
- `C:/windows/system32/dllcache/fltmgr.sys`
- `C:/windows/system32/ntoskrnl.exe`
- `C:/windows/system32/wbem/Repository/FS/OBJECTS.MAP`
- `C:/windows/system32/wbem/Logs/FrameWork.log`

---

**New Files**

- `C:/windows/system32/config/systemprofile/Application Data/Microsoft/CryptnetUrlCache/Content/60E31627FDA0A46932B0E5948949F2A5`
- `C:/windows/system32/config/systemprofile/Application Data/Microsoft/CryptnetUrlCache/MetaData/60E31627FDA0E46932B0E5948949F2A5`
C:/windows/system32/fltmc.exe
C:/windows/system32/wbem/AutoRecover/679BDCDA90EDF24EC83945D38F2EEEAA.mof
C:/windows/system32/CatRoot/[F750E6C3-38EE-11D1-85E5-00C04FC295EE]/oem5.CAT
C:/windows/system32/drivers/MpFilter.sys
C:/windows/system32/LegitCheckControl.dll
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/3C83474D61E624A4F9844DF935AFE217
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/904590238400AD963F77FAAAADC9BAB5
C:/windows/system32/CatRoot/[F750E6C3-38EE-11D1-85E5-00C04FC295EE]/oem4.CAT
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/904590238400AD963F77FAAAADC9BAB5
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/303572DF538EDD8B1D606185F1D559B8
C:/windows/system32/CatRoot/[F750E6C3-38EE-11D1-85E5-00C04FC295EE]/KB914811.cat
C:/windows/system32/wbem/AutoRecover/E1DEF7F9703AD28DD4B87F9536940147.mof
C:/windows/system32/drivers/vmnet.sys
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/A8FABA189DB7D25FBA7CAC806625FD30
C:/windows/system32/drivers/vmnetadapter.sys
C:/windows/system32/libeay32.dll
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/3C83474D61E624A4F9844DF935AFE217
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/79841F8EF00FBA86D33CC5A47696F165
C:/windows/system32/vnetinst.dll
C:/windows/system32/CatRoot/[F750E6C3-38EE-11D1-85E5-00C04FC295EE]/WGA.cat
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/79841F8EF00FBA86D33CC5A47696F165
C:/windows/system32/drivers/vmnetadapter.sys
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/303572DF538EDD8B1D606185F1D559B8
C:/windows/system32/CatRoot/[F750E6C3-38EE-11D1-85E5-00C04FC295EE]/oem3.CAT
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/A8FABA189DB7D25FBA7CAC806625FD30
C:/windows/system32/wbem/AutoRecover/D0A684ACFF6A24ECEC199D3475E1BF1C.mof
C:/windows/system32/ssleay32.dll
C:/windows/system32/Logfiles/W3SVC1/ex060328.log
C:/windows/system32/wbem/AutoRecover/84970C837E25CB34EEE411E6F5A3DFFC.mof

---

**Missing Files**

C:/windows/system32/drivers/fltMngr.sys
C:/windows/system32/fltMc.exe
C:/windows/system32/LegitCheckControl.DLL

---

**Process ended at Mon Apr 3 10:27:32 2006**

---

Total Runtime: 213 Seconds.
Altered Files
C:/windows/system32/config/AppEvent.Evt
C:/windows/system32/wbem/Logs/FrameWork.log
C:/windows/system32/spmsg.dll
C:/windows/system32/ntkrnlpa.exe
C:/windows/system32/fltlbib.dll
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp
C:/windows/system32/wbem/Repository/FS/MAPPING2.MAP
C:/windows/system32/mapisvc.inf
C:/windows/system32/MsDtc/Trace/dtctrace.log
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/catdb
C:/windows/system32/config/SysEvent.Evt
C:/windows/system32/wbem/Logswmofcomp.log
C:/windows/system32/CatRoot2/{127D0A1D-4EF2-11D1-8608-00C04FC295EE}/catdb
C:/windows/system32/wpa.dbf
C:/windows/system32/config/systemprofile/ntuser.dat
C:/windows/system32/inetsrv/MSDTC.LOG
C:/windows/system32/wbem/Logs/wmipro.p.log
C:/windows/system32/CatRoot2/edb.log
C:/windows/system32/wbem/Logs/wbemprox.log
C:/windows/system32/wbem/Repository/FS/INDEX.BTR
C:/windows/system32/ntoskrnl.exe
C:/windows/system32/config/SecEvent.Evt
C:/windows/system32/wbem/Repository/FS/OBJECTS.MAP
C:/windows/system32/perfc009.dat
C:/windows/system32/CatRoot2/edb.chk
C:/windows/system32/wbem/Logs/wbemess.log
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/E6024EAC88E6B6165D49FE3C95ADD735
C:/windows/system32/MSDTC.LOG
C:/windows/system32/wbem/Repository/FS/MAPPING1.MAP
C:/windows/system32/CatRoot2/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/TimeStamp
C:/windows/system32/dllcache/fltlbib.dll
C:/windows/system32/wbem/Logs/wmiadap.log
C:/windows/system32/perfh009.dat
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/E6024EAC88E6B6165D49FE3C95ADD735
C:/windows/system32/dllcache/ftlmgr.sys
C:/windows/system32/config/systemprofile/ntuser.dat.LOG
C:/windows/system32/CatRoot2/dberr.txt
C:/windows/system32/wbem/Repository/FS/OBJECTS.DATA
C:/windows/system32/dllcache/ftlmc.exe
C:/windows/system32/wbem/Repository/FS/INDEX.MAP
C:/windows/system32/wbem/Logs/FrameWork.log
C:/windows/system32/wbem/Logs/wbemess.log
New Files
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/60E31627FDA0A46932B0E5948949F2A5
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/60E31627FDA0A46932B0E5948949F2A5
C:/windows/system32/fltmc.exe
C:/windows/system32/spool/drivers/w32x86/3/HP6nlcrk.cfg
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/oem5.CAT
C:/windows/system32/drivers/MpFilter.sys
C:/windows/system32/LegitCheckControl.dll
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/3C83474D61E624A4F9844DF935AFE217
C:/windows/system32/spool/drivers/w32x86/3/HPj4ta9l.cfg
C:/windows/system32/spool/drivers/w32x86/3/HPFDJ845.GPD
C:/windows/system32/spool/drivers/w32x86/3/HPFDJ84X.GPD
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/Content/904590238400AD963F77FAAAADC9BAB5
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/oem4.CAT
C:/windows/system32/spool/drivers/w32x86/3/HPFDJ200.HLP
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/KB914811.cat
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/WGA.cat
C:/windows/system32/CatRoot/{F750E6C3-38EE-11D1-85E5-00C04FC295EE}/oem3.CAT
C:/windows/system32/config/systemprofile/Application
Data/Microsoft/CryptnetUrlCache/MetaData/A8FABA189DB7D25FBA7CAC806625FD30
C:/windows/system32/spool/drivers/w32x86/3/HPFNAM50.GPD
C:/windows/system32/wbem/AutoRecover/D0A684ACFF6A24ECEC199D3475E1BF1C.mof
C:/windows/system32/spool/drivers/w32x86/3/HPFIMG50.DLL
C:/windows/system32/spool/drivers/w32x86/3/HPFIMG50.DLL
C:/windows/system32/logfiles/W3SVC1/ex060328.log
C:/windows/system32/wbem/AutoRecover/84970C837E25CB34EEE411E6F5A3DFFC.mof

---

**Missing Files**

C:/windows/system32/drivers/ftlmgr.sys
C:/windows/system32/ftlmc.exe
C:/windows/system32/legitcheckcontrol.dll

---

**Process ended at Tue Apr 11 14:07:10 2006**

---

Total Runtime: 227 Seconds.