San Jose State University

SJSU ScholarWorks

Summer 2010

UNDO: A System for Neutralizing Nuisance Attacks

Eilbroun Waltin Benjamin
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

UNDO: A SYSTEM FOR NEUTRALIZING NUISANCE ATTACKS

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Eilbroun W. Benjamin

August 2010

The Designated Thesis Committee Approves the Thesis Titled

UNDO: A SYSTEM FOR NEUTRALIZING NUISANCE ATTACKS

by

Eilbroun W. Benjamin

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

August 2010

Dr. Mark Stamp	Department of Computer Science
Dr. Michael Beeson	Department of Computer Science
Dorothy McKinney	Lockheed Martin Corporation

ABSTRACT

UNDO: A SYSTEM FOR NEUTRALIZING NUISANCE ATTACKS

by Eilbroun W. Benjamin

In recent years, our society has seen a shift towards a reliance on digital means of data storage. This thesis considers the problem of digital data integrity protection, which is defined as preventing unauthorized writing of data. Numerous examples of successful attacks against seemingly secure targets are examined to support the assertion of the author that, at least in some circumstances, the integrity of digital data is difficult to preserve.

An approach to securing data is proposed in which a security administrator first assumes that a system will be compromised. This approach limits its focus to a nuisance-type attack, which is defined as an attempt to obscure shared non-sensitive data by limited-experience attackers. A trusted third party, the Universal Nuisance Defense Object (UNDO), is employed to monitor the system and automatically detect and abate unauthorized writing of data. This approach is further expanded upon by utilizing a tool set of metrics that allows one to measure the performance of UNDO and appropriately configure it. This allows an administrator to optimize its efficiency, ideally to the point where this category of attack on the data integrity will be nullified.

ACKNOWLEDGEMENTS

Table of Contents

List of Figures

List of Tables

1. The Need for Digital Data Integrity Protection

In the past decade, our society has developed a dependence on various digital systems to perform many routine functions. These functions can range from looking up published information on a company or an organization to significantly more sensitive tasks such as managing banking accounts and investments [1]. A shift to digital media increases the need for digital files and databases to make this data available. Using digital media to find, analyze, and manipulate data creates a need for maintaining the integrity of this data. In the context of cryptography, data integrity is defined to be the prevention of unauthorized writing of data [2]. One wants to be able to trust the published company information they look up at the company's website, and certainly to trust the online banking data associated with his or her account.

The Internet provides an example of this shift. Consumers expect sufficient online access to their accounts to manage them. This access opens the possibilities for online hacking [3]. Even the most sensitive and critical online data such as government protected secrets and online banking have been compromised at times [1]. If the institutions that understand the rigid security requirements of digital data storage are becoming victims, how can average users hope to protect their systems and their data?

Some individuals and groups even assist in the proliferation of hacking by offering hacking tools and expertise to the general public. For example, a flaw was discovered with the IBM 4758 crypto processor, and it was published with the title

"Hack Your Bank for $995" [4]. This article points interested readers to a how-to guide

for repeating the attack. The common crypto processor in question is responsible for

accepting a user's PIN number, securely transmitting it to a central location, and

verifying its validity. The architecture of the system is thought to be secure, but the

software that runs on it, the Common Cryptographic Architecture (CCA), has a flaw.

Researchers Michael Bond and Richard Clayton were able to find a method to have the

crypto processor send them its encryption key [4]. Although this encryption key was

encrypted with a different crypto key, the second encryption was found to be easier to

crack. With this encryption key, one can retrieve every account number and PIN that is

entered into the particular machine.

When this flaw was discovered and shared, it opened many companies up to

vulnerability because of the common use of this mechanism. To make matters worse,

the flaw was associated specifically with the free software that came with the unit. In

order to respond, many different companies needed to implement their own costly

fixes, which would vary in execution time and effectiveness. Some companies may have

opted to take no action if they perceived the flaw as an unlikely threat.

There is documentation that even the software systems that support the United

States government agencies, such as the Pentagon, the National Aeronautics and Space

Administration (NASA), and the various branches of the armed forces have been

compromised in the past. Gary McKinnon, known also as SOLO, is being extradited to

the United States from the United Kingdom to face charges of implementing what a

prosecutor describes as "the biggest military computer hack of all time" [5]. It is alleged that over the course of two years, McKinnon was able to gain unauthorized access to ninety-seven computer systems owned and operated by various United States government agencies.

McKinnon was able to successfully implement hacks on what are thought of as highly secure systems by implementing simple, well-known hacking methods that any person can look up and learn. He analyzed the computer networks to determine which areas to focus his attack. He then determined what software was in place and researched associated vulnerabilities. And finally, he exploited these vulnerabilities to gain access to seemingly secure systems.

The research and attack methods McKinnon used generally do not require highly sophisticated tools or advanced expertise. For example, network mapping software is readily available. With this software, one can analyze a network and determine the topology [1]. Even without any special software, one can simply employ the ping and traceroute utilities to see how the network topology is laid out. The path a packet travels through the network reveals this information.

Another reconnaissance step is to scan a system for known vulnerabilities. There are common tools that scan the open ports of a particular computer, check the software running on a system, and so forth. This information is then used to determine vulnerabilities [1]. Based on the software versions that are on a system, the attacker may find various weaknesses to focus on. Once the vulnerabilities are found, the

attacker can use spoofing to hide their identity and implement attacks to exploit the vulnerabilities.

The examples thus far have involved highly skilled professionals targeting sensitive data. There is, however, a more familiar type of attack which one may liken to vandalism. This involves an attacker manipulating data presented by a legitimate source, by either falsifying it or removing it altogether. The author of this thesis was consulted by a non-profit organization whose website had recently been hacked. The attacker gained access to the system, deleted all system data, and left only the website frame containing his signature. This website was an important communication tool linking the organization with the general public. Maintaining a credible, trustworthy, and secure image was crucial, since it relied on public support for its efforts. The organization feared that public perception of its legitimacy had been adversely affected by this act of online vandalism.

This project focuses on protecting the integrity of data, which may be a document, an email, a program, or any other digital communication form. All of these examples are stored as data on a disk. Depending on the operating system or the database management system (DBMS), the raw data can be organized and accessed as a file containing data, or a database entry. Maintaining the integrity of this data is crucial in many applications. The examples above illustrate how easy it is to attack a system, and also suggest that it is difficult to defend a system from attack.

2. An Approach to Ensuring Digital Data Integrity

Common implementations of file systems or databases rely on well-known technologies. The problem that arises from this is that well-known technologies often have well-known security weaknesses associated with them [1]. As described in the introduction, someone attempting to compromise the integrity of the system can research this information and exploit these vulnerabilities. One possible solution is to design a new data sharing implementation from the ground up. A major problem with this approach is that it is difficult to design secure systems [3]. The security level of these custom applications will not have been proven by review from a wide audience, leading to the possibility that an attacker will be able to analyze them and find a weakness, possibly with less effort than in the first case.

The inherent difficulty in designing a secure system stems from the conflict between functionality and security [6]. An increase in functionality will generally lead to an increase in system complexity. An increase in complexity will generally lead to more opportunities for vulnerabilities to occur. Ordinarily, a system is not designed solely to be secure; rather, it is designed first and foremost to provide functionality. That is, the security level of a particular data sharing system may be a highly regarded feature of the system, but it will never be the sole intention of the system. Continuous changes and updates to this functionality often lead to highly complex systems that are difficult to secure. Each update potentially opens the door to new vulnerability issues.

The purpose of this thesis is to propose a new approach to increase the level of

integrity protection on digital data storage implementations when dealing with

nuisance-type attacks. We define a nuisance-type attack as an attempt to obscure non-

sensitive shared data by limited-experience attackers. A primary goal is to increase the

level of integrity protection by changing the focus of these nuisance-type attacks. This is

accomplished by introducing a new tool, the Universal Nuisance Defense Object

(UNDO), which will work independently from other security measures. The design of

UNDO is based on the assumption that an attacker will be able to compromise a

protected system and make unauthorized modifications to the data. UNDO can mitigate

this by detecting unauthorized modifications and taking action.

There are existing commercial products that attempt to address the issue of data

integrity protection. They generally use a cryptographic hash function to compute and

store an authorized state of the data, and continuously compare this with the current

state of the data. A typical product offering this service is Ionx's Data Sentinel [7]. This

product is advertised as a highly advanced Host Based Intrusion Detection System

(HIDS). The user of this system chooses when to take a "snapshot," at which time the

software analyses the system and stores what it deems to be critical system settings

that include file data and attributes. When users decide to check the integrity of the

system data, they use the software to run a comparison of the current system state and

produce a report for the user. The company Runtimeware produces a product named

Sentinel, which works similarly to the Ionx's Data Sentinel, except that it also monitors

operating system data such as the registry for signs of unauthorized activity [8].

The approach in this thesis is similar to the commercial products mentioned above in that UNDO is responsible for data integrity issues. However, UNDO is different in that it takes an active role in fixing such issues. UNDO acts as a trusted third party (TTP), and will not only have read access to the system being protected, but also have write access. This TTP will maintain the integrity of the system data by monitoring it, detecting unauthorized changes, and resetting parts of the system to their last authorized state as necessary. Another difference in this approach is that the TTP allows diminutive modifications to data by one or more authorized users. A proactive pattern recognition and encryption scheme is employed to determine if a user's modification command is authorized. The general design of this approach is depicted in Figure 1.

Figure 1. TTP integrity system design.

This approach creates two general TTP vulnerabilities. The first is that an attacker may be able to compromise the protected system and trick the TTP into thinking nothing is wrong. The second is that if an attacker compromises the TTP, then

he or she will be able to use it to manipulate the protected system. From the basic design that is laid out, it becomes apparent than one must compromise this TTP in some manner in order to successfully compromise the protected system, which is the target. However, since the scope of the TTP is limited, it is theoretically easier to have a highly secure design and implementation. The goal of this design is to force an attacker to defeat the higher level security of the TTP, regardless of the vulnerabilities of the actual system. This supposedly makes the protected system as secure as the TTP. A second goal is to have the protected system maintain a security level equivalent to the TTP, regardless of security weaknesses that may arise as a result of system updates.

To clarify this goal, assume we have an accurate security level measurement for a protected system on a scale that ranges from zero to 100. This security rating is based on the level of effort required to compromise the system. A highly secure system will require a high level of effort while a less secure system will require less. A rating of zero indicates that compromising the protected system is effortless while a rating of 100 indicates that compromising the protected system is a daunting task. The author is careful to state "daunting task" instead of "impossible task" since successful attacks over the years have shown that it is not realistic to expect a system to be completely secure. Suppose the level of effort required to compromise the protected system is measured at 30, and also suppose the level of effort to compromise the TTP is measured at 80 on the same scale. Figure 2 depicts the level of effort required to compromise each system.

Level of Security Comparison

Figure 2. Level of effort to compromise an unprotected system.

Initially, an attacker will likely focus their efforts on the less secure protected system. Yet, as stated above, a goal of this system is to force the attacker to compromise the TTP before the protected system. This implies that compromising the protected system will require the same amount of effort as compromising the TTP, as depicted in Figure 3.

Level of Security Comparison

Figure 3. Implied level of effort to compromise a protected system.

How can a reactive security implementation be expected to significantly increase the level of integrity in a digital data storage system? An attack can be made essentially

pointless by having the reactive security measure quickly identify and "undo" unauthorized changes, which is the primary goal of the system. The protected system will be restored within a predefined time that correlates with the system's guaranteed up-time goals.

For example, consider the non-profit organization mentioned in the first section. Their website was constructed with an organized file structure and no actual database. For simplicity, suppose their goal is to achieve an up time of 99%, and an ability to handle up to one attack per 24-hour time period. Also, let us assume there is a negligible data restoration time when an unauthorized change is identified. Suppose we implement this security scheme and set it to check the entire digital data system every 15 minutes. If an attacker successfully makes a single modification to the file system, then the security scheme will detect the attack and act within 15 minutes. This implies a maximum system data down time of approximately 1% during a 24-hour time period, assuming only a single attack is implemented during that period.

3. General UNDO System Architecture

This section describes the architecture of UNDO, which is the functional tool that implements the security scheme described in the introductory sections. The functionality of UNDO is described as well as its interaction with a protected system. In addition, noted limitations are discussed. The implementation of UNDO is a highly customizable tool that tracks a series of protected resources, which can be either files or database tables. It also allows minor modifications to already established protected resources.

3.1 UNDO User Types

There are three types of users considered by UNDO: an UNDO administrator, an UNDO user, and a protected system user. The UNDO administrator has access to the UNDO machine and carries credentials required to change system settings and manage UNDO users. An UNDO user is able to make diminutive modifications to protected resources, and carries a set of credentials required to make such changes. A protected system user can be any individual that accesses the protected system's resources. UNDO does not attempt to manage protected system users in any way.

3.2 Housing UNDO

The first design point to consider is housing UNDO, which depends on the circumstances surrounding its implementation. One option is to house all UNDO components on the same machine as the system being protected. The performance

testing done in support of this thesis as described in Section 4 clearly identifies data

transfer as the slow point in operations. This would be an issue for congested networks

since there would be an additional transfer delay as data is moved through the network.

However, it is difficult to prove one system is any more secure than another when they

are living on the same machine. This machine would likely have many vulnerability-

inducing features enabled. Moreover, having UNDO live on the same machine can

actually create a new vulnerability because it is essentially a single point of failure for

protected data integrity.

For these reasons, UNDO was implemented on a dedicated machine separate

from the protected system. The protected system is free to be as flexible as it needs

while the machine hosting UNDO is able to impose more rigid security requirements.

This distributed approach creates the need for a secure communication channel

between UNDO and the protected system. This was achieved by implementing a Secure

Socket Layer (SSL) connection over an Ethernet network as depicted in Figure 4.

Figure 4. UNDO housing diagram.

Note that although the UNDO-dedicated machine and the protected system have an

established network connection, the network they share is closed. The protected

system has a separate connection to the outside world, thereby exposing it to attackers.

3.3 UNDO System Components

We now turn our focus to how UNDO provides integrity protection. The

interface of the protected system with UNDO was described in the introduction, but it is

necessary to go one step further to describe the interface among UNDO components:

the UNDO User Interface, the UNDO Command Processor, the UNDO Backup Database,

the UNDO SSL Connector, and the UNDO Protected System (PS) SSL Connector. Figure 5

shows the system flow between components, and the following sections describe how

the pieces work together.

Figure 5. UNDO system flow diagram.

The UNDO User Interface is responsible for allowing an UNDO administrator to

view and modify system settings. This component also displays system status and

activity reports to an administrator. The UNDO Command Processor is the central point

where all UNDO actions are either executed or approved. This segment listens for and

executes commands from the UNDO User Interface including adjusting system settings,

as well as managing and verifying protected resources. Another function of this
segment is to listen for modification commands from UNDO users and verify these
against system pattern recognition and encryption settings. The UNDO Backup
Database stores all the data necessary to allow system functionality. This primarily
includes the backups of protected resources, but certain system settings and
administrator reporting data are stored here as well.

The final piece that resides on the UNDO machine is the UNDO SSL Connector,
which enables two-way communication between the UNDO machine and the protected
system. On the other end of this connection resides the UNDO PS SSL Connector, which
is the only component of UNDO that resides on the protected system. The UNDO PS SSL
Connector not only receives commands from the UNDO Command Processor through
the UNDO SSL Connector, but it also is responsible executing them. The UNDO PS SSL
Connector also listens for UNDO user modification commands, which are sent to the
UNDO Command Processor for verification before execution.

3.4 UNDO Functionality

This section concerns the functionality provided by the UNDO components
working in tandem with each other. The primary responsibly of the system is to provide
integrity protection for protected resources. As such, an UNDO administrator is able to
specify new resources on a protected system to guard. If the resource is a file, the
administrator must only specify the file location on the protected system and the PL. If
the resource is a database table, the administrator must specify the table name,

database type, database location, database credentials, and the PL. Figure 6 shows the

display for protecting new resources.

Figure 6. UNDO protect resource display.

Administrators must also be able to manage protected resources. They are able

to change resource PLs and unprotect resources. In order to do this, an administrator

must find the resource in the resources display and use the PL drop-down box or

unprotect button. This is depicted in Figure 7 below.

Figure 7. UNDO manage protected resource display.

From the same display shown in Figure 7, the administrator can choose to

immediately verify a particular resource by clicking on the verify button. The other

option is to enable the auto-verification feature, which will automatically choose

protected resources to verify depending on UNDO's configuration. For automatic

integrity protection, an administrator controls three basic components. The first is the

interval window, which has been described earlier as the administrator's definition of

"real-time" in terms of the protected system. The administrator can also change the file

to database ratio, which essentially specifies how much of the computing resources are

designated for one or the other. Another ratio to be configured is the priority level,

which is described in Section 3.5. Figure 8 shows the administrator's configuration

screen.

Figure 8. UNDO integrity configuration display.

The UNDO system is also responsible for accepting modification commands from

authorized UNDO users and processing them. The UNDO administrator selects whether

these commands will not be verified, verified only by pattern recognition, verified only

by encryption, or verified by both pattern recognition and encryption. Figure 9 shows

the administrator's display.

Figure 9. UNDO modification command configuration display.

The UNDO users are responsible for knowing the configuration and following the appropriate protocols as described in Section 7. The current configuration of UNDO only allows user to send SQL insert into commands that are 256 characters long. The format for a modification command is outlined in Figure 10.

Figure 10. UNDO modification command format.

In the case of pattern recognition being enabled, the UNDO administrator also specifies the delta length and delta margin. These are used to configure the rigidity of pattern recognition to compensate for various network environments. These are discussed in greater detail in Section 7 as well. Also depicted in the figure above is a threshold value, which once reached, will cause the system to stop processing UNDO user commands. Finally, an UNDO administrator is able to manage UNDO users, by adding new ones, deleting existing ones, and changing existing ones' shared secret keys. This function is depicted in Figure 11.

Figure 11. UNDO user management display.

3.5 Resources and Priority Levels

A resource protected by UNDO can be either a file or a database table. A multilevel classification scheme is introduced to give certain resources higher protection standards than others. Because the protected system can be made up of any number of resources, the user needs to be able to classify each resource with a Priority Level (PL). The PLs considered in this thesis are listed in Table 1:

Table 1. Possible Priority Levels (PLs).

PL0	no integrity protection
PL1	low integrity protection
PL2	standard integrity protection
PL3	high integrity protection
PL4	real time integrity protection

An administrator provides a definition of "real-time" in terms of the protected system by specifying the length of the interval window. This interval window size is essentially a refresh rate for PL4 resources, specifying the maximum amount of time a PL4 resource can go without verification. Each PL4 is verified once at the beginning of each interval. PL1, PL2, and PL3 resources are verified with the remaining computation

18

time within the interval. An administrator specifies the priority among PL1, PL2, and PL3

resources, presumably giving PL3 resources the highest priority, followed by PL2, and

finally by PL1. This is done by assigning each of these three PL levels a percentage such

that the sum of the three percentages is equal to 100%. Each PL percentage signifies

the portion of the remaining computation time of the interval to be used for that

particular PL.

3.6 UNDO Command Processing

UNDO needs to be a dynamic real-time system, and as such, a three-level

interrupt-based scheme for command processing is used to achieve this capability.

UNDO uses different interrupt levels for different commands and has lower level

interrupt commands yield to higher level interrupt commands. Processing administrator

commands is a level 3 interrupt, while auto-verifying PL4 resources is defined as a level

2 interrupt, and auto-verifying any other PL level resource is defined as a level 1

interrupt. When the auto-verification feature is enabled, UNDO would continue to

monitor PL1, PL2, and PL3 resources until an administrator command is issued or the

interval window ends and it becomes time to review PL4 resources. If an administrator

command is issued, that command is immediately processed. If it becomes time to

auto-verify PL4 resources, and there are no administrator commands waiting to be

processed, UNDO will halt the process of monitoring PL level 1 to 3 resources until all

PL4 resources are verified. This interrupt logic is depicted in Figure 12 as a flow

diagram.

Figure 12. Command processing interrupt logic flow.

3.7 Detected Issue Resolution

There are two types of attacks to be considered by UNDO. The first is

unauthorized modifications made directly to protected resources. This type of attack

can be detected by UNDO when a particular resource is verified. The UNDO Command

Processor executes the verification by requesting the hash of the resource from the

UNDO PS SSL Connector and verifying this against the stored hash in the UNDO Backup

Database. Once a discrepancy is discovered, the UNDO Command Processor executes a

quarantine procedure to create a copy of the unauthorized data modification. The

UNDO Command Processor then replaces the unauthorized data on the protected

system with the backup it has stored in the UNDO Backup Database. A report of any

UNDO attack detection activity is logged and displayed for administrator review.

The second type of attack is unauthorized modifications made to protected resources through UNDO by passing UNDO User commands without the appropriate permissions. The UNDO administrator specifies whether there is no authentication, pattern recognition only, encryption only, or a combination of pattern recognition and encryption. An authorized UNDO user is expected to know these settings and send appropriately processed modification commands. A command can fail pattern recognition by being sent at a wrong time. A command can fail encryption by not being encrypted or being encrypted by the wrong key. A command that fails either or both of these security schemes is considered a single UNDO user command failure. The administrator sets a failure threshold and once this threshold is surpassed, UNDO stops processing UNDO user commands. Any accepted or failed UNDO user command is logged for UNDO administrator review.

3.8 Security Concerns

A primary security concern arises from this basic design in that an attacker can use UNDO to manipulate the protected system by compromising UNDO and using it to issue commands. This is abated by minimizing the portion of UNDO that resides on the protected system. The UNDO PS SSL Connector is the only portion lives on the protected system. It is stateless, not storing any access control information for protected resources including usernames and passwords. It relies on the UNDO Command Processor sending this information when necessary and forgets the information as soon as it is used. The UNDO PS SSL Connector has no special privileges

on the protected system, so even after compromising it, an attacker will not gain privileges. The UNDO Command Processor lives on a separate machine that is configured for rigid security. An attacker would need to first access the protected system, and then access the machine hosting UNDO through the established secure connection.

The other main security concern is an attacker compromising the UNDO PS SSL Connector to trick UNDO into thinking an unauthorized data state is okay. The UNDO PS SSL Connector would have to be modified to send bad information back to the UNDO machine. This is tricky because it would need to comply with the established protocols for processing UNDO commands, or an UNDO administrator will see the system performance has halted. In addition, the UNDO PS SSL Connector needs to send back hashes that the UNDO Command Processor expects, which may not be valid for the actual protected resources. This portion of UNDO has the highest security settings available on the protected system. Although this portion of UNDO is the most susceptible to attack, defeating it will be no easier than defeating the operating system security. Nothing will be gained in addition to possibly preventing UNDO from performing its duties.

4. Efficiently Detecting Changes in Data

A primary concern of this thesis is efficiency, and at the heart of this concern is the process by which changes in data are detected. As it is important to analyze different approaches, and a naïve comparison was first implemented to act as a baseline. From there, other possible approaches were researched and compared against it. One approach that will be explored in this thesis is the computation of a hash, which has been found to be an effective solution in many applications.

4.1 A Naïve Comparison Approach

A naïve approach to this problem is to back up the data and do a byte-by-byte comparison. Considering unpredictable data sizes and memory constraints, a fixed length buffer could be used to read in blocks of data for comparison purposes. Suppose data is stored in a source file and a difference check is to be conducted. Prior to performing this check, it is assumed that a backup of the source file exists to act as a baseline for future comparisons. Figure 13 shows the pseudo code for this naïve algorithm approach.

```
const int BUFFERSIZE = 1024;
byte [BUFFERSIZE] bufferSource   // source data buffer
byte [BUFFERSIZE] bufferBackup   // backup data buffer
while (more data in file 1) {
   bufferSource = next 1024 bytes of source file
   bufferBackup = next 1024 bytes of backup file
   if (bufferSource != bufferBackup)
      return false   // stop, a difference has been found
}
return true    // all data has been checked
```

Figure 13. Naïve comparison algorithm pseudocode.

4.2 Naïve Comparison Analysis

The size of the input data is unpredictable and can vary greatly. One comparison may consist of 50-byte entries in a database field while another comparison consists of one-gigabyte files. As a result, the efficiency test was designed to span a wide range of likely data sizes. This was a worst-case scenario test in which each pair of data points being compared was identical. The test consisted of 32 rounds, each of which employed different sized collections of data for comparison. Each round used a file composed of ASCII characters ranging in size from zero bytes to one gigabyte. The first round used a zero-byte data file. The second round used a one-byte data file, and each proceeding round used a data file that was double the size of the previous round. The file data size for each round is depicted in Figure 14 below.

Figure 14. Data size for each round.

Note that the time analysis of the Naïve Compare algorithm can be described as

24

$2n$, where n is the number of bits read in. This value is derived as follows: for every file

consisting of n bits, n bits need to be read in from the source file, and n bits need to be

read in from the backup file to be a basis of comparison. Testing analysis demonstrated

that once the data have been loaded into main memory, the time for the comparison is

negligible compared to the time to load it in. The results from testing the Naïve

Compare algorithm are depicted in Figure 15.

Figure 15. Naïve Compare algorithm time analysis results.

The measured processing time of the Naïve Compare algorithm behaves in a

similar manner to the data growth rate depicted in the efficiency test layout (Figure 14).

This is expected considering the time analysis of the Naïve Compare algorithm and the

growth rate of the data from round to round, which indicate the computation time

should double in each round. Table 2 lists the actual processing times. The times for

the first 17 tests were negligible as the data sizes are diminutive compared to the buffer

size.

Table 2. Naïve comparison algorithm time analysis results.

Size (B)	Time (ms)	Size (B)	Time (ms)
0.00E+00	0	3.28E+04	0
1.00E+00	0	6.55E+04	1
2.00E+00	0	1.31E+05	2
4.00E+00	0	2.62E+05	3
8.00E+00	0	5.24E+05	6
1.60E+01	0	1.05E+06	120
3.20E+01	0	2.10E+06	169
6.40E+01	0	4.19E+06	231
1.28E+02	0	8.39E+06	543
2.56E+02	0	1.68E+07	1,132
5.12E+02	0	3.36E+07	4,253
1.02E+03	0	6.71E+07	11,417
2.05E+03	0	1.34E+08	20,866
4.10E+03	0	2.68E+08	34,736
8.19E+03	0	5.37E+08	53,931
1.64E+04	0	1.07E+09	88,957

4.3 A Cryptographic Hash Approach

An alternative comparison approach is to compute a hash of the data using a

cryptographic one-way hash algorithm. The Message Digest 5 (MD5) Hash function is

one such example that is widely used. An input of any length will always produce a 128-

bit output [9]. The algorithm begins by breaking the input data into blocks of 512 bytes

and padding the last block if it is less than 512 bytes. For each block, a series of "and,"

"or," "exclusive or," and shifting operations are performed following the protocol

described in the design document. Once the input data has produced a 128-bit output,

this output can be compared to a stored 128-bit hash to verify the data is similar.

Following the pigeonhole principle, it is apparent that this algorithm can create the same output for multiple inputs, which is a situation referred to as a collision. MD5 Hash has the property where similar inputs produce different outputs, which works to prevent meaningful attacks [10]. A meaningful attack is described as an attack where two or more different inputs, neither of which is filled with obscure data, produce the same output. This can be made more difficult by also checking the size of the data being compared.

An attacker can, however, find junk data that happens to hash to the same output, and replace the existing data to trash it and making it impossible for an MD5 Hash comparison to detect the change [11]. This can be considered a limitation for the automation of UNDO, as it creates the possibility that junk data may be introduced into the system and remain undetected by UNDO. However, this is not a concern at this time, as it is not currently feasible to be given a hash value and find another value that hashes to it. In any case, junk data will be easy for any user to identify making this a denial of service issue where the actual data may not be available to the user rather than a data integrity issue where a user retrieves false information from the protected system.

4.4 MD5 Hash Algorithm Analysis

The run time analysis of the MD5 Hash algorithm would lead one to expect a smaller computation time when compared to Naïve Compare. Because MD5 only needs to read in n bits of data, the average time analysis for computing a hash is described as

n. Once again, the time to transfer data from secondary storage to main memory

nullifies the time to process the algorithm once data is in main memory. An additional

step is added for comparing the hash against another stored hash, which is run in full

regardless of whether or not the data is equal. This creates an average run time of $n +$

128. Testing results are depicted in Figure 16.

Figure 16. MD5 Hash algorithm time analysis results.

Once again, the measured processing time of the MD5 Hash algorithm behaves

in a similar manner to the data growth rate in the efficiency test layout (Figure 14).

Given the growth rate of the data from round to round and the time analysis, the

computation time should double each round. Table 3 lists the actual processing times

for this test. Once again, the computation times for the first seventeen rounds were

negligible due to diminutive data size.

Table 3. MD5 hash algorithm time analysis results.

Size (B)	Time (ms)	Size (B)	Time (ms)
0.00E+00	0	3.28E+04	0
1.00E+00	0	6.55E+04	1
2.00E+00	0	1.31E+05	2
4.00E+00	0	2.62E+05	3
8.00E+00	0	5.24E+05	2
1.60E+01	0	1.05E+06	8
3.20E+01	0	2.10E+06	18
6.40E+01	0	4.19E+06	35
1.28E+02	0	8.39E+06	70
2.56E+02	0	1.68E+07	215
5.12E+02	0	3.36E+07	1,140
1.02E+03	0	6.71E+07	2,234
2.05E+03	0	1.34E+08	5,425
4.10E+03	0	2.68E+08	8,362
8.19E+03	0	5.37E+08	18,182
1.64E+04	0	1.07E+09	35,570

4.5 Comparison of Change Detection Approaches

Having compared the computation times of these two algorithms, two important

factors are noted. First, the MD5 Hash algorithm can make a comparison faster than the

naïve comparison by a factor of nearly three. This may appear surprising, but the

difference can be explained in the implementation of the algorithms. As shown above,

the MD5 hash algorithm reads n + 128 bits for an n-bit long data stream. The Naïve

Compare algorithm, however, reads $2n$ bits for an n-bit long data stream because it is

comparing two identical streams of data. Reading in large amounts of data from

secondary storage becomes the bottleneck as a result. The comparison of both

algorithms is depicted in Figure 17.

Computation Time Comparison

Figure 17. Time comparison of MD5 Hash and Naïve Compare algorithms.

The second factor to notice is the relatively negligible computation time for both algorithms when the data size is up to 32,768 bytes. This is significant because the MD5 hash algorithm produces an output of 128 bits, which is then compared to a stored 128 bit hash. This shows that the additional step does not increase the computation time by a noticeable amount. The time analysis for MD5 Hash, which was described as $n + 128$ earlier, can now be described as n.

5. Ensuring File Integrity

This section concerns file-specific details for integrity protection. UNDO has focused on two distinct methods of data storage: files and databases. For the purposes of this thesis, a file is simply a collection of related data stored in secondary storage. The operating system decides how to store and retrieve each file. The data change detection scheme, described in the previous section, can be applied in a specific manner to achieve the file protection goals of the system implementation.

5.1 File Priority Levels

Due to limited computing resources, a priority classification system for data was introduced in Section 3 that assigned each protected resource a PL. Each individual file is considered a protected resource, and as such, is assigned its own PL. Possible PL values can range from 0 to 4. A PL of 0 indicates that no integrity protection is warranted. A PL of 1 indicates minimum priority, 2 indicates standard priority, 3 indicates high priority, and 4 indicates real-time priority. A file with a PL of 0 will never be checked while a file with a PL of 4 will be checked once during each interval.

5.2 UNDO File Storage Analysis

This procedure also has storage constraints that need to be analyzed. In order to efficiently detect changes, a hash is needed, which consists of a fixed length of 128 bits. A backup copy of the original file is needed to replace the shared file when corruption is detected. For a file consisting of n bits, an additional n bits are needed to back it up.

Also, a pointer to the original file's location is needed so the system can retrieve it. As a result, the number of bits the TTP Integrity System will need to store for each file being backed up is $\theta(128 + n + m)$, where n is the number of bits in the file and m is the number of bits to represent the file's location. The size of the file will dominate the size of the hash and the size of the pointer to the file, thereby making the storage space constraints $\theta(n)$.

6. Ensuring Database Integrity

Larger or more complex systems must rely on databases in addition to their file systems to store and access their considerable amount of data. Many systems use a combination of file systems and databases to store all the required data. This makes it vital to monitor and preserve the integrity of the data stored in a database as well. This section describes the UNDO method for maintaining database integrity.

6.1 DBMS Disk Management

Even though data may be stored in a relational database, the database still resides in some type of storage device. As long as the system knows the address of the data, it can be treated the same as file data discussed in the MD5 Hash section earlier. The problem with this approach is that most Database Management Systems (DBMS) move data at their own discretion to optimize efficient access to the data, especially when the constraints are redefined. In this case, the integrity system would need to be updated and this would cause an increased level of complexity. As a result, UNDO needs to access data through the DBMS.

6.2 Standard Data Integrity Approaches

Preserving the integrity of the data in a database is a common practice in standard implementations of databases that do not take provisions to protect against malicious attackers. There are a number of approaches under the family of Redundant Array of Independent Disks (RAID) [12]. In RAID level 1, a disk containing the database

information is backed up by a redundant disk. If the first disk fails, the information is

likely to be maintained in the second disk. In this approach, one would need one entire

redundant disk for every disk of data. In an improvement, RAID level 4 uses only one

redundant disk for any number of information disks. The modulo-2 sum of the bits in

each information disk is calculated and stored in the redundant disk. This makes it

possible to recover from a single disk error. RAID level 6 can guarantee recovery from a

certain number of disk crashes, depending on the number of redundant disks in the

system.

Another example is use of the binary checksum, where certain columns in the

particular tuple are looked at and others are ignored [12]. The modulo-2 sum of the bits

that make up the meaningful data that is looked at in this example is computed, stored,

and used to check for errors later. The problem with this approach, along with the more

general RAID approaches, is that they are designed to prevent, detect, and correct bit

errors caused by machine malfunction. They are not designed to cope with an

intelligent selection of data to corrupt. For example, an attacker may know that the

binary checksum approach is used. They can purposely make changes so that the

modulu-2 sum of their changes always equals the original value. This approach would

then fail to detect a compromise in integrity.

6.3 DBMS Based Integrity

An approach that is resistant to malicious attackers is using the hash of the data

to determine if there are data modifications. Rather than treating it as data on a disk,

the data are accessed through the DBMS as tuples organized into tables. An example of

this is to take the hash of each row of data within that table such as in Figure 18. Taking

the hash of an entire table is not realistic as a table is likely to change often, which

would mean the hash of the entire table would need to be computed for even minor

changes. Let us say we have a table that contains one gigabyte of data, and a single

byte of data is changed at some point within the table. The existing hash of the table is

now invalid, and the hash algorithm would need to be computed once again for the

entire table, such as in Figure 19.

Figure 18. Hash of entire table example.

Figure 19. Hash of entire table with 1-bit change.

The previous approach can be refined such that a hash is computed for each row

of data in a particular table, such as in Figure 20. Now, making a minor change to data

within the table will result not in a re-computation of the hash for the entire table, but

rather just that particular row as depicted in Figure 21. The issue that arises with this

approach is that an individual hash comparison would need to be computed for each

row in a database table, resulting in a longer computation time for each entry into the

database. The next section takes this analysis into account and describes the actual

approach used for UNDO.

TABLE 1	
ROW 1 DATA	ROW 1 HASH
ROW 2 DATA	ROW 2 HASH
ROW 3 DATA	ROW 3 HASH

Figure 20. Hash of each row example.

TABLE 1	
ROW 1 DATA	ROW 1 HASH
ROW 2 DATA	ROW 2 HASH
ROW 3 DATA	ROW 3 HASH

Figure 21. Hash of each row with 1-bit change.

6.4 Database Hierarchical Hash

Incremental hash is a process in which a single hash is maintained for a

presumably large data structure. The purpose of this method is to have a situation

where a small update to the data results in a level of effort for re-computing the hash that is proportional to the change [13]. The first approach in Section 6.2 does not meet this goal since a small change to one row of data will result in the entire table being hashed again. Unfortunately, an incremental hash approach is not appropriate for UNDO, as the incremental hash relies on knowing the difference between the updated data and the original. However, a Hierarchical Hash (HH) is defined in this thesis to allow UNDO to more efficiently monitor and maintain database data.

A single HH is specified for each protected table. Each HH consists of data with equal PL and is grouped under the same PL. Once a PL is chosen for inspection, each table will have equal probability for being selected for verification. In this approach, a significant amount of data can be verified by comparing a single hash. When a new row is introduced to a protected table, the hash for the row is computed and stored, and the HH is computed to be the hash of all the row hashes. Once a new table is introduced to UNDO, each row will be hashed and an HH will be computed by taking the hash of the hashes of each row. A table with an HH is depicted in Figure 22 below.

TABLE 1	
ROW 1 DATA	ROW 1 HASH
ROW 2 DATA	ROW 2 HASH
ROW 3 DATA	ROW 3 HASH
TABLE 1 HIERARCHICAL HASH (HH)	

Figure 22. HH example.

7. Pattern Recognition

A dog is waiting by the window for his beloved owner to come home. There is a seemingly infinite number of ways the owner can choose to do so: parking in different places, following different paths, moving at various speeds, and so forth. However, the dog knows that the owner parks a large truck next to the tree across from the window and quickly walks up the side path towards the garage, tripping on the same displaced piece of concrete each time, and enters the house through the side door. The way in which the owner arrives at the home can be described as a pattern. Depending upon its complexity, a pattern can be more or less difficult for others to mimic. This complexity provides a baseline for the dog to determine if the person in question is, in fact, the owner, and if excitement is warranted.

An implementation of pattern recognition is applied to how data is entered into a database protected by UNDO. Each UNDO user has a distinct pattern that no other can easily mimic. A pattern is created based on the time UNDO receives a modification command from the UNDO user. The authorized UNDO user will send commands to insert new data into the database at appropriate times that are easy to verify by UNDO, but difficult to predict by an attacker. There is an obvious tradeoff here in that the more detailed the pattern, the harder it is to mimic; however, the more detailed the pattern, the more time consuming and difficult it is to verify as well.

7.1 Pattern Recognition Implementation

When an UNDO user sends commands to modify a protected system, the collection of times at which each individual command is sent forms a pattern. That pattern can be used to verify if the commands came from the authorized user. For this, a delta value is defined as the least significant n digits of the command received time displayed in milliseconds, where n is specified by the UNDO administrator. A series of these deltas forms the pattern. UNDO and the UNDO users agree on a method to determine future delta values. Each delta needs to appear unpredictable and offer enough variation. If any portion of the pattern is predictable then an attacker can mimic the predicted time algorithm exactly. If there is not enough variation in the time, then the attacker can guess the next time with high success probability. For example, if the next time in the sequence can only be one of two choices, the attacker has a fifty percent change of guessing correctly.

Having the user and system agree on delta values is accomplished by using the common cryptography method of using a shared secret key. The UNDO PS SSL Connector listens for modification commands from an UNDO user. Each UNDO user has a shared key with UNDO. Both UNDO and the UNDO user independently use the key to determine the next delta for that particular user. The UNDO administrator imposes restrictions on the delta by setting the delta length and delta margin values. The delta length is the number of base-10 digits the delta is composed of, and the delta margin is the number if milliseconds a delta can be varied from the expected delta value and still

be acceptable.

An UNDO user first sends an initialize message to the UNDO PS SSL Connector, which will then respond with its current system time. The UNDO user uses this to account for system time differences, and can repeat this process multiple times to find a more accurate difference. The UNDO user implementation repeats the initialize process three times and uses the average as the initial system time. This initial system time will be used to synchronize system times and determine the initial delta as well. The protocol is shown in Figure 23 below.

Figure 23. Initial user synchronization with TTP integrity system.

The initial time is combined with the 128-bit shared secret key to form a 192-bit string of data. This string is run through the MD5 Hash algorithm to produce a 128-bit output. The delta value is extracted from the hash result, depending on the delta length setting imposed by an UNDO administrator. In the implementation, a delta length can be between 2 and 4, providing a range of between 10 and 9,999 milliseconds. A delta length of 4 requires 14 bits from the hash, but a delta length of 3 requires only 10 bits, and a delta length of 2 requires 7 bits. Once the initial delta has been computed, future deltas can be determined by running the current delta through the MD5 hash algorithm along with the secret key, and extracting the necessary bits from the result.
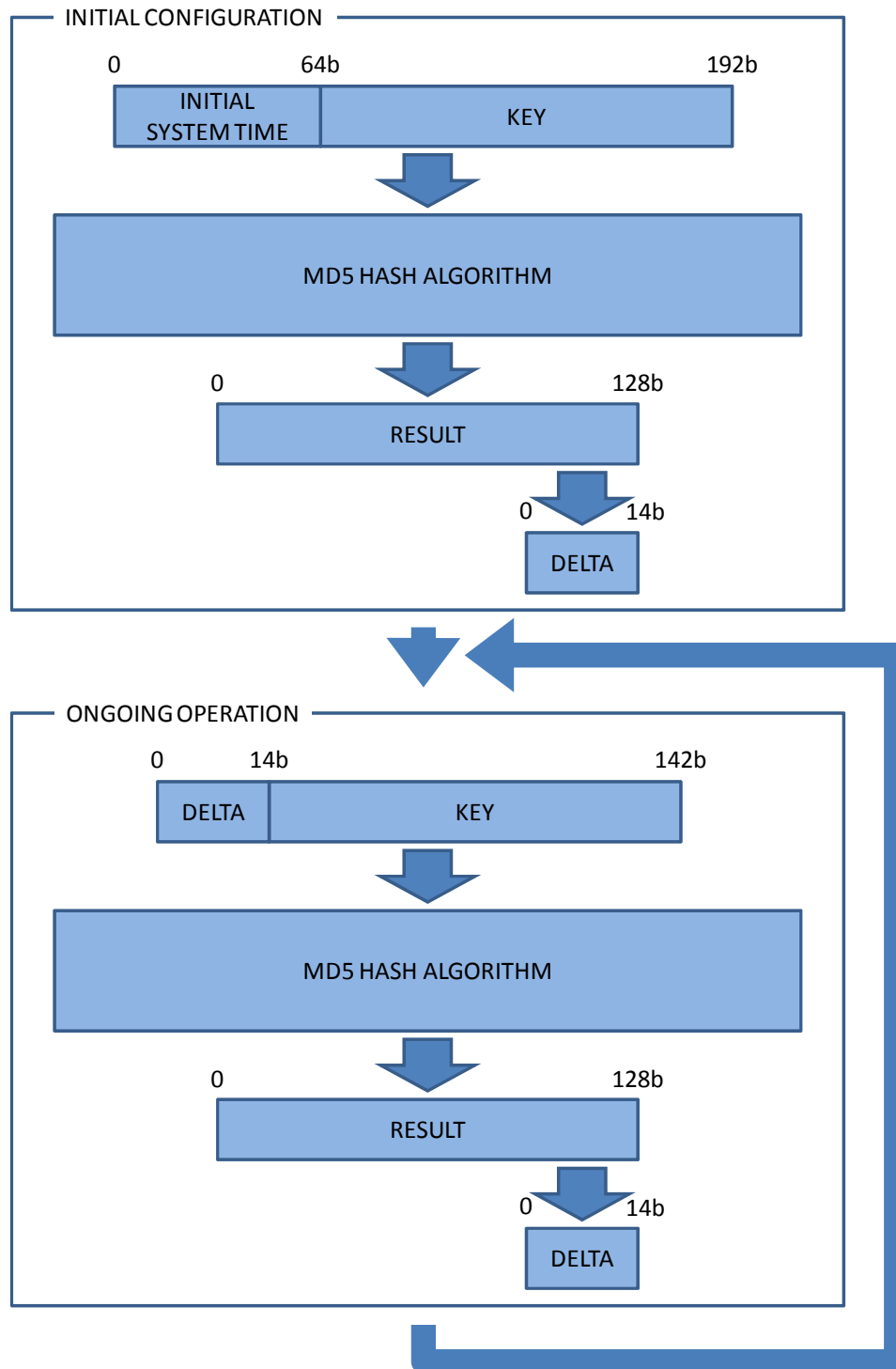
The algorithm flow is shown in Figure 24.

Figure 24. Delta computation algorithm flow.

The implementation described herein is based on a 128-bit key. The fastest way to accurately find the next predicted time at any point should be by guessing every possible secret key value, which would provide an expected level of security directly related to the size of the key. A large enough key will make it impractical to determine the next predicted time through any means other than using the secret key.

7.2 Security Analysis

Because dynamic databases require frequent updates, the predicted time must be accurate to the most precise level possible. Otherwise, an entry that is expected to be entered in milliseconds will need to wait in a queue for extended periods of time. UNDO cannot predict when the next modification command will arrive, so it is only concerned with the seconds and milliseconds of a command received time. It is theoretically possible to make this algorithm accurate to the microsecond, but operating system differences and network latency makes this infeasible in implementation. A delta length of 4 provides a range of zero milliseconds to 9,999 milliseconds, which allows each delta to be chosen from 10,000 possibilities.

In order for an attacker to mimic the UNDO pattern, a naïve approach would be to attempt to correctly guess the next delta, which has a probability of up to:

$$\frac{1}{10,000}$$

or 0.01%. For an attacker to correctly guess any 2 deltas, the probability is up to:

$$\left(\frac{1}{10,000}\right)^2$$

or 0.000001%. For an attacker to correctly guess any n deltas, the probability is:

$$\left(\frac{1}{10,000}\right)^n$$

Taking into account the delta length, d_l, and the delta margin, d_m, the UNDO

implementation has the following probability for an attacker being able to correctly

guess any n delta values:

$$\left(\frac{1}{10^{d_l} - 2 \cdot d_m}\right)^n$$

It appears that an attacker cannot expect to defeat the system by randomly

guessing the delta values. An alternative is adaptively choosing delta values. This would

be done by analyzing the current delta values and finding a pattern in the calculations.

A hashing function was chosen to compute the deltas specifically for this reason. Figure

25 shows an example with 1,000 sequential deltas that were computed with this

method. It is obvious from this figure that the hash function behaves in an

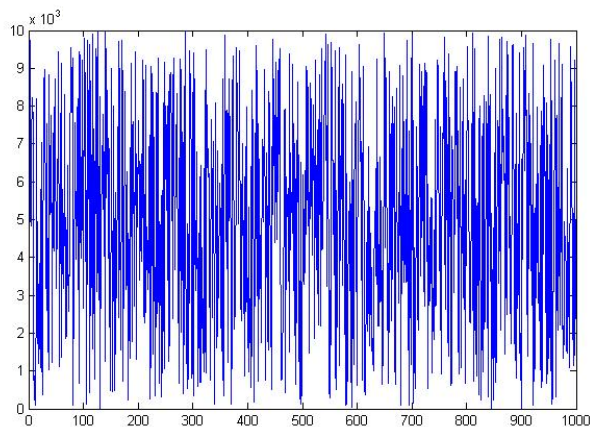unpredictable manner, making trend analysis an arduous task.

Figure 25. Graph of 1,000 sequential deltas.

Another approach that is always available for an attacker is guessing the 128-bit

secret key. If an attacker has a supercomputer that can process one key every

nanosecond, then it can process 1,000,000,000 keys every second. An attacker would

expect to find the key using this brute force method in $1/2 * 2^{128}/1,000,000,000$

seconds, or 1.701e+29 years, giving the UNDO administrator plenty of time to change

the key.

7.3 Advantage of this Approach

A more obscure advantage to this implementation leverages the longer time it

takes to send and approve protected system modification commands. This runs

contrary to the efficiency goals elsewhere in this thesis, but may be applicable to a

situation where an administrator wants to minimize changes to protected data. Take

the case of a protected system with tremendous volumes of data that is not regularly

modified. Based on the delta length value identified by the UNDO administrator, the

time it would take an attacker to corrupt a substantial portion of the protected data can

be determined. The UNDO administrator can purposely set a larger delta length to slow

down an attack. An attacker that sends commands quickly will be detected immediately

by having delta values that are too small. By conforming to the system rules to avoid

immediate detection, the attacker will spread apart his or her attack. Knowing this, the

UNDO administrator can set a particular minimum delta value such that if the attacker

attempts to spread their data upload to conform, it will take too long for the attack to

be considered serious to the system, or give the UNDO administrator time to react. A
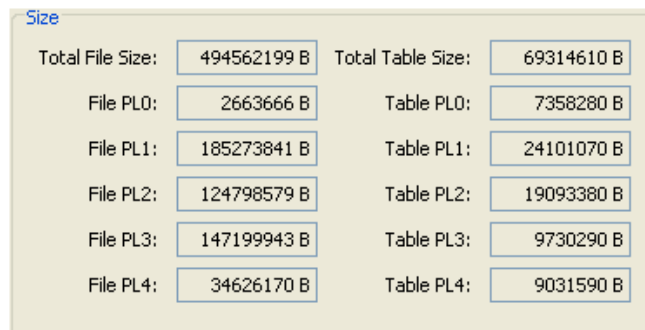
larger delta length value also minimizes processing time devoted to changes, which

allows the system to use more resources for integrity issue detection and resolution.

8. UNDO Configuration

The general overview discussed the goals of this project in maintaining an efficient UNDO system. An UNDO administrator is able to configure the system and attain an expected level of performance. This section details how the UNDO administrator is able to measure the performance of UNDO. These metrics are available through the UNDO User Interface. Some of these metrics are indirectly controlled by the administrator by modifying system settings while others are based directly on the administrator's priorities.

8.1 Protected Resource Sizes

There are two types of protected resources: files and tables. The size display shows the total size of all protected resources broken down first by whether they are a file or table. The second breakdown displays the total size of each PL for each type of protected resource. All sizes are in bytes. Figure 26 shows the UNDO display.

Size			
Total File Size:	494562199 B	Total Table Size:	69314610 B
File PL0:	2663666 B	Table PL0:	7358280 B
File PL1:	185273841 B	Table PL1:	24101070 B
File PL2:	124798579 B	Table PL2:	19093380 B
File PL3:	147199943 B	Table PL3:	9730290 B
File PL4:	34626170 B	Table PL4:	9031590 B

Figure 26. UNDO protected resource size display.

These metrics are tracked by variables for other calculations. The following table lists

the variables and their purpose.

Table 4. UNDO protected resource size names.

Variable Name	Variable Description
s_{file_total}	total size of protected files
s_{file_pl0}	size of pl0 files
s_{file_pl1}	size of pl1 files
s_{file_pl2}	size of pl2 files
s_{file_pl3}	size of pl3 files
s_{file_pl4}	size of pl4 files
s_{table_total}	total size of protected tables
s_{table_pl0}	size of pl0 tables
s_{table_pl1}	size of pl1 tables
s_{table_pl2}	size of pl2 tables
s_{table_pl3}	size of pl3 tables
s_{table_pl4}	size of pl4 tables

8.2 Verification Rates and Open Interval Time

All protected resources that are files are verified with the same file verification

process despite their PL. The only difference is the frequency of this verification based

on their PL. In the same manner, all protected resources that are tables are verified

with the same table verification process despite their PL. The only difference is, once

again, the frequency of this verification. However, there is a difference in how a file is

verified as opposed to a table, so distinct verification rates for the two are required. The

UNDO Command Processor measures the time to make each verification and updates

the rate variables with the verification time and size of data. The verification rate for

files is stored in v_{file} and the verification rate for tables is stored in v_{table}. Similarly, there

is also a need to find the verification time for all PL4 data, including both files and tables,

which is stored in variable t_{pl4}. This in measured is milliseconds and is used to determine

the open interval time, which is simply the interval time specified by the UNDO

administrator minus the PL4 verification time, and stored in variable t_{open}. An open

interval time of zero indicates there is no time to verify non-PL4 data. Figure 27 shows
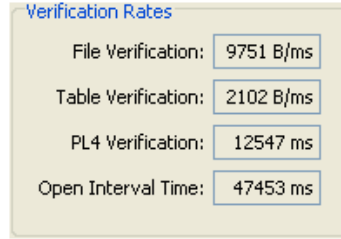
the screenshot for this display.

Figure 27. UNDO verification rates display.

8.3 PL1, PL2, and PL3 Verification per Interval

The PL1, PL2, and PL3 verification metrics display the percentage of each

respective PL group to be verified during one interval. These are once again broken

down by not only resource type, but PL as well. The administrator defined probability

for files is p_{file}, and for tables is p_{table}. The administrator defined PL probabilities are p_{pl1},

p_{pl2}, and p_{pl3}. These verification metrics are computed as follows:

$$p_{file\_pl1} = t_{open} \cdot p_{file} \cdot p_{pl1} \cdot \frac{v_{file}}{s_{file\_pl1}}$$

$$p_{file\_pl2} = t_{open} \cdot p_{file} \cdot p_{pl2} \cdot \frac{v_{file}}{s_{file\_pl2}}$$

$$p_{file\_pl3} = t_{open} \cdot p_{file} \cdot p_{pl3} \cdot \frac{v_{file}}{s_{file\_pl3}}$$

$$p_{table\_pl1} = t_{open} \cdot p_{table} \cdot p_{pl1} \cdot \frac{v_{table}}{s_{table\_pl1}}$$

$$p_{table\_pl2} = t_{open} \cdot p_{table} \cdot p_{pl2} \cdot \frac{v_{table}}{s_{table\_pl2}}$$

$$p_{table\_pl3} = t_{open} \cdot p_{table} \cdot p_{pl3} \cdot \frac{v_{table}}{s_{table\_pl3}}$$

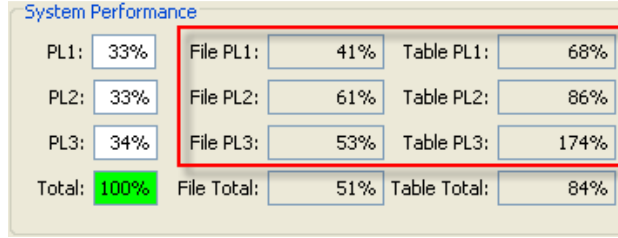The following figure shows the UNDO display for these values.

Figure 28. PL1, PL2, and PL3 interval verification percentages display.

8.4 UNDO Administrator Importance Metric

The UNDO administrator can use a custom metric to quickly determine if UNDO is configured to focus on the appropriate priorities. The administrator will specify an importance value for PL1, PL2, and PL3 resources, respectively. The importance value for PL1 resources is i_{pl1}, for PL2 is i_{pl2}, and for PL3 is i_{pl3}. This value will be a percentage such that the sum of the three values equals 100%. The importance values will act as a weight differentiating the PLs. Leveraging the values computed from 8.3, the following equations are used to determine the file and table importance metrics respectively:

$$i_{file} = i_{pl1} \cdot max(p_{file\_pl1}, 1) + i_{pl2} \cdot max(p_{file\_pl2}, 1) + i_{pl3} \cdot max(p_{file\_pl3}, 1)$$

$$i_{table} = i_{pl1} \cdot max(p_{table+pl1}, 1) + i_{pl2} \cdot max(p_{table\_pl2}, 1) + i_{pl3} \cdot max(p_{table\_pl3}, 1)$$

Considering the weighted importance values assigned, the administrator can determine if the current configuration is appropriate for either protected files or tables. If the administrator is concerned heavily for PL3 resources, but cares little about PL2 or PL1 resources, they can assign PL3 an importance of 80%, and PL1 and PL2 an

49

importance of 10% each. Then the percentage of PL3 data verified in a single interval will weigh heavily for the i_{file} and i_{table} values while PL1 and PL2 data will have minimal impact. The UNDO display for these metrics is shown in Figure 29.
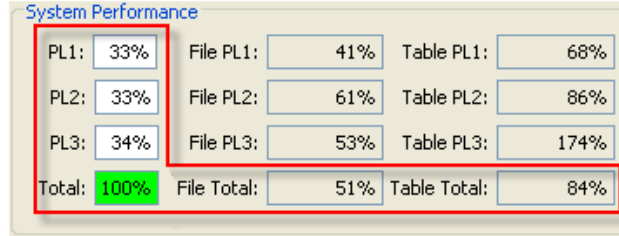

Figure 29. UNDO administrator PL importance display.

8.5 UNDO Administrator Run Time Metrics

Another metric is implemented to take the interval window into account along with other system configurations. An UNDO administrator inputs a run time, t_{run}, in minutes. The system first determines the percentage of total PL4 data to be verified in that time, represented by r_{pl4}. This value can only be a multiple of 100%. If the run time is long enough to check PL4 data, but less than the interval time, then it is 100%. If the run time is long enough for exactly one interval to complete, then it is 100% since PL4 data will be checked once. If one interval can be completed and a portion of a second interval where all PL4 data can be verified, then it is 200%.

The expected percentage of file and table data classified as PL1, PL2, and PL3 to be verified is computed as well. This is done by taking the total open interval time available after PL4 data has been verified during the run time, then finding the amount of time distributed between both files and tables as well as between PL1, PL2, and PL3

data. This is approximated by multiplying the total open interval time by the percentage

assigned to file data, or table data, then the percentage assigned to a particular PL.

$$r_{file\_pl1} = \left(t_{run} - \left(r_{pl4} \cdot t_{pl4}\right)\right) \cdot p_{file} \cdot p_{pl1} \cdot \frac{v_{file}}{s_{file\_pl1}}$$

$$r_{file\_pl2} = \left(t_{run} - \left(r_{pl4} \cdot t_{pl4}\right)\right) \cdot p_{file} \cdot p_{pl2} \cdot \frac{v_{file}}{s_{file\_pl2}}$$

$$r_{file\_pl3} = \left(t_{run} - \left(r_{pl4} \cdot t_{pl4}\right)\right) \cdot p_{file} \cdot p_{pl3} \cdot \frac{v_{file}}{s_{file\_pl3}}$$

$$r_{table\_pl1} = \left(t_{run} - \left(r_{pl4} \cdot t_{pl4}\right)\right) \cdot p_{table} \cdot p_{pl1} \cdot \frac{v_{table}}{s_{table\_pl1}}$$

$$r_{table\_pl2} = \left(t_{run} - \left(r_{pl4} \cdot t_{pl4}\right)\right) \cdot p_{table} \cdot p_{pl2} \cdot \frac{v_{table}}{s_{table\_pl2}}$$

$$r_{table\_pl3} = \left(t_{run} - \left(r_{pl4} \cdot t_{pl4}\right)\right) \cdot p_{table} \cdot p_{pl3} \cdot \frac{v_{table}}{s_{table\_pl3}}$$

The following figure shows the display for this metric.

Figure 30. UNDO administrator run time metric display.

9. UNDO Testing Results

UNDO system-level testing was completed to show the effectiveness of UNDO in providing integrity protection by reactive data verification and authorized modification processing. Various tests were designed to measure the efficiency of the system and its resistance to attack. Protected resources were simulated by 563,876,809 bytes of generated data. Of this, 1% was PL0, 31% was PL1, 29% was PL2, 30% was PL3, and 8% was PL4.

9.1 Integrity Protection Testing

The integrity protection testing was concerned with attackers modifying protected resources through some means other than compromising and using UNDO. An attacker is simulated by using the protected system's OS and DBMS to make direct modifications to protected data. The auto-verify feature of UNDO was then enabled with the specified configuration for each test.

9.1.1 Standard Attack Test

The purpose of the standard attack test was to simulate the type of attack UNDO was designed to handle and to monitor its response. In this test, standard UNDO system configuration was used so that the file and table ratio was left at 50% and 50%, respectively. The priority level configuration was left at PL1=33%, PL2=33%, and PL4=34%. The interval window was set to 1 minute to compensate for the smaller amount of data. An attacker modified a total of 60 resources, 30 of which were file

resources and the other 30, table resources. The 30 compromised file resources were

evenly distributed among the priority levels 1 through 3, and the same was done for

table resources. Figure 31 shows the UNDO administrator's display as the system was

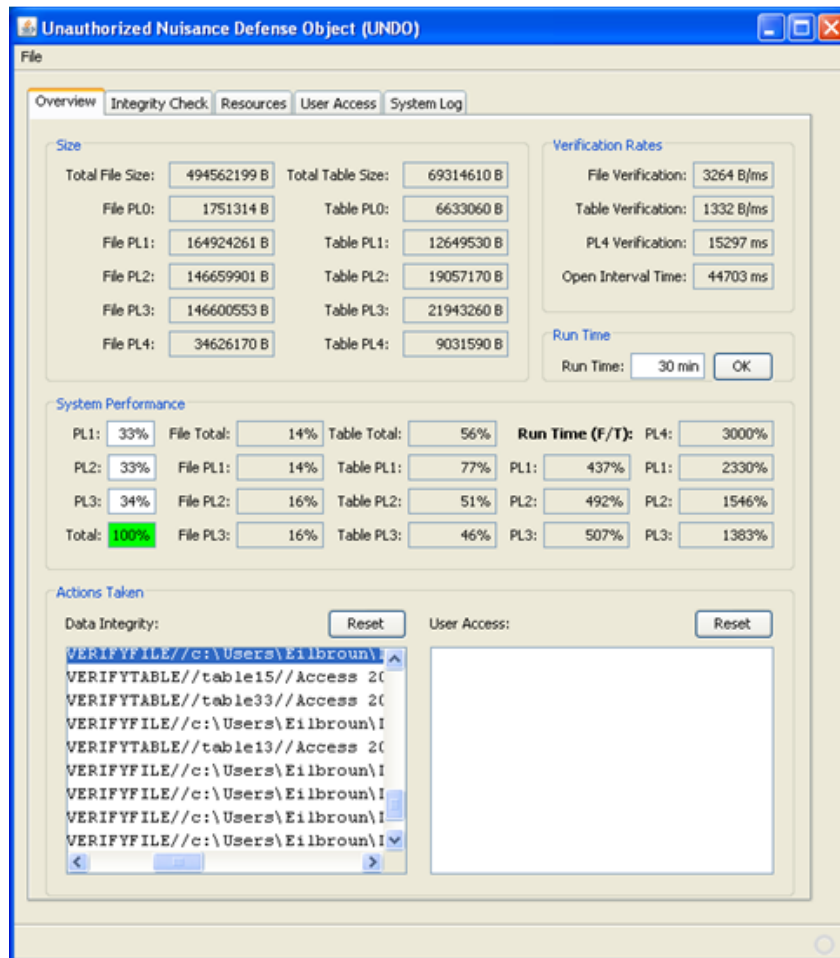detecting and fixing compromised data.

Figure 31. UNDO administrator display during attack.

Note that the smaller amount of table data made it more likely to verify those

resources, even though the verification rate was less than half of the file verification

rate. In an actual implementation, this display would notify the UNDO administrator to

reconfigure the file and table ratio for more balanced verification.   Table 1 shows the

attack recovery times for all data. The recovery time is the time it took for all of the

resources falling in the resource type category to be checked and restored.

Table 5. Standard attack recovery

Resource Type	Recovery Time
Table PL3	221 seconds
Table PL1	263 seconds
Table PL2	497 seconds
File PL2	1,030 seconds
File PL1	1,165 seconds
File PL3f	1,277 seconds

9.1.2 Full Recovery Test

The full recovery test considered 500 protected resources ranging from PL1 to

PL4. PL1 resources constituted 33% of the data while PL2 and PL3 resources each made

up 30% of the data, and PL4 resources were 7%. All protected data was modified at the

beginning of the test, and the number of resources fixed throughout the test was

tracked by PL. Figure 32 displays the results.

Figure 32. Full recovery test with PL1=33%, PL2=33%, and PL3=35%.

The first minute of the test lead to approximately 225 resources being restored, which is expected since the vast majority of resources being verified at that time were corrupted. As time went forward, less data was verified as more time was used to re-verify already fixed resources due to the randomized nature of the algorithm. All PL2 through PL4 data was verified within 9 minutes leaving only 5 PL1s. A full system recovery occurred in 10 minutes and 20 seconds.

A second test was conducted where the priority level ratio was shifted to focus on PL3 resources first, then PL2 resources, and finally PL1 resources. This caused the full recovery time to increase to 11 minutes and 1 second. Figure 33 shows the results of this test.

Figure 33. Full recovery test with PL1=20%, PL2=30%, and PL3=50%.

All PL3 data was verified within 6 minutes, which was an improvement over the 8 minutes of the previous test. PL2 data was verified within another minute. After 7 minutes of running tests, the only resources not fixed were 14 PL1s.

Even more resources were devoted to PL3 data in the following test. This left

PL1 data with only 10%, which stretched the test out to 34 minutes and 56 seconds. This change in prioritization caused all PL3 data except to 2 resources to be fixed within the 3 minutes. The 2 remaining resources were reset in the next minute. Figure 34 displays UNDO activity.
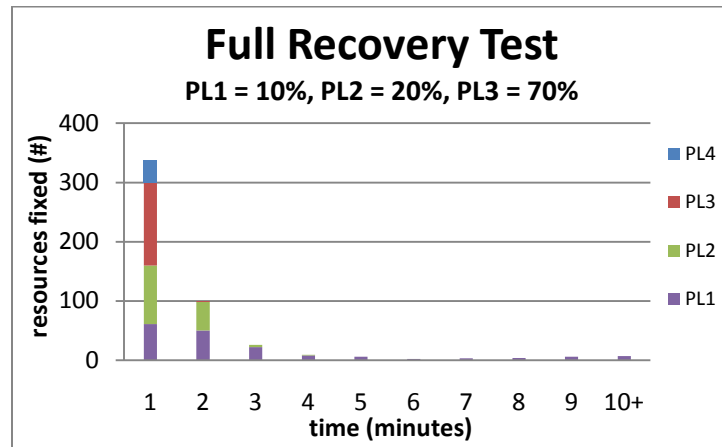

Figure 34. Full recovery test with PL1=10%, PL2=20%, and PL3=70%.

The final test gave PL3 resources 100% of the open interval time, but it still took about 4 minutes to restore all PL4 data. This was due to the randomized nature of the algorithm in choosing which resource to verify next. Figure 35 displays the results.

Figure 35. Full recovery test with PL1=0%, PL2=0%, and PL3=100%.

9.1.3 Consistent Attack Test

The concern of concerted, distributed attacks was identified early in the thesis. One such attack was simulated over the course of 7 minutes to see if system response was adequate. In the first minute, all 500 protected resources were available. The attack then struck once at the beginning of each of the next 5 minutes. Each attack wave deleted all protected resources. Over the course of this attack, an average of only 33% of the protected resources were available. Figure 36 shows the attack results.

Figure 36. Example of consistent attack on UNDO.

9.2 UNDO User Command Testing

UNDO user command testing was concerned with UNDO's ability to accept authorized modification commands to protected resources. An UNDO user was created to simulate a user that was aware of the protocols and could provide valid encryption and pattern recognition commands. This same UNDO user was used to mimic an attacker that knew the system protocols but did not have possession of the secret key.

9.2.1 Baseline Test

The baseline test consisted of 1,000 UNDO User commands being sent to UNDO with pattern recognition and encryption disabled. The purpose of this test was to provide an efficiency baseline for the other authentication mechanisms. Each individual test run started with a valid insert command being sent by an UNDO User, and ended with a response being received by the UNDO User. It took 33,671 milliseconds to successfully send 1,000 commands and receive responses, as shown in Figure 37.
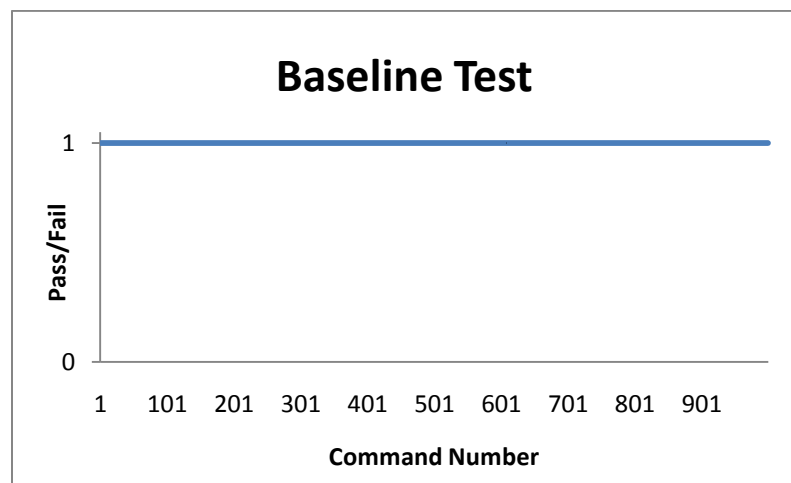

Figure 37. Baseline pass and failure success graph.

9.2.2 Encrypted Commands Test

The encryption test consisted of 1,000 encrypted UNDO User commands being sent by an UNDO User with encryption enabled but pattern recognition disabled. The UNDO User was in possession of a valid secret key. As in the baseline test, there was a 100% success rate as depicted in Figure 38 below. The duration of the test was 313,906 milliseconds, nearly 10 times longer than the baseline test.
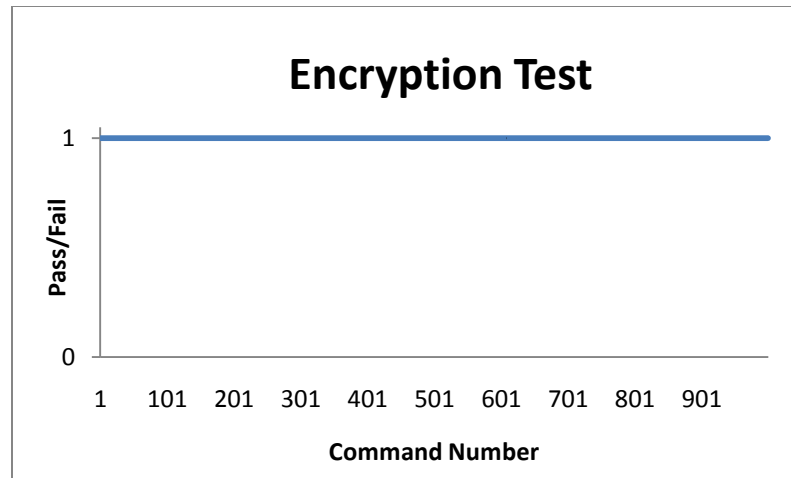
Encryption Test

Figure 38. Encryption pass and failure graph.

9.2.3 Pattern Recognition Commands Test

This series of tests monitored the effects of pattern recognition configuration on the UNDO system. As such, pattern recognition was enabled but encryption disabled. The UNDO administrator is able to configure the delta length, d_l, and delta margin, d_m, for pattern recognition. Recalling from Section 7, the d_l is the number of base-10 digits that form the delta, and the d_m is the error margin that compensates for network latency. Three separate tests were implemented, the first of which used a d_l of 3 and d_m of 25. This test lasted 422,578 milliseconds. The configuration gave an attacker a 1/950 chance of guessing a correct delta value. Note that the delta was re-initialized in the middle of the test, which explains why the series of failures towards the middle of Figure 39 immediately stop. Depending on the d_m value, the UNDO user and UNDO were found to move out of sync as a test continues. This can be alleviated by re-initialing the delta after a certain amount of time has passed.
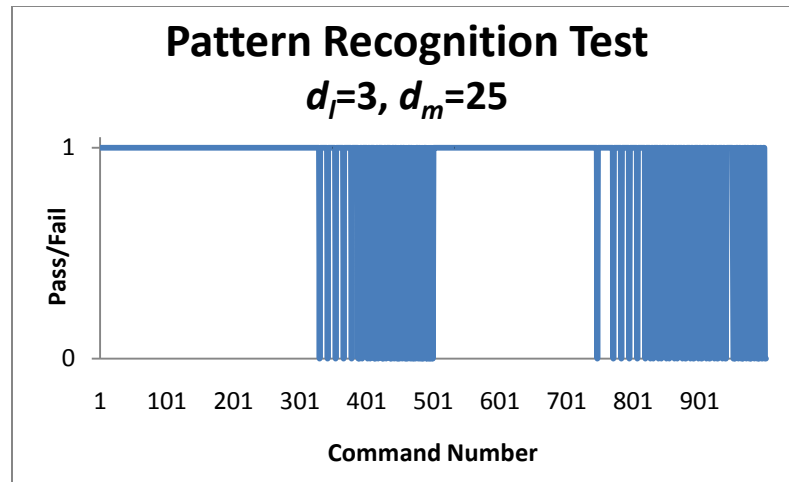
59

Figure 39. Pattern recognition pass and failure graph with d_l = 3, d_m = 25.

The second test used a d_l of 3 and delta margin of 15, and lasted 433,140 milliseconds. This gave an attacker a 1/970 chance of successfully guessing a single delta value, a slight security improvement over the previous test.
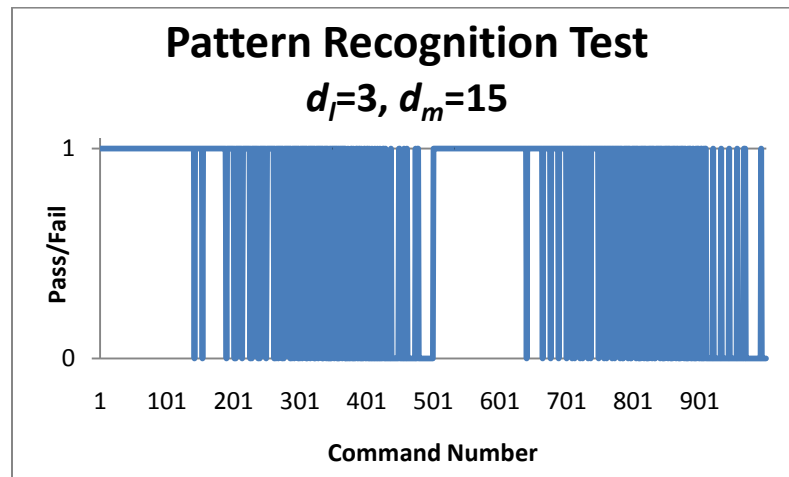

Figure 40. Pattern recognition pass and failure graph with d_l = 3, d_m = 15.

Note that this is only a 2% security improvement over the previous configuration, but the delta times come out of sync substantially sooner in the test.

The third test used a d_l of 2 and d_m of 15, giving an attack a notably improved

1/2 chance of successfully guessing a single delta value. Having a smaller d_l allowed the UNDO user send commands faster. The test lasted only 167,219 milliseconds, approximately 60% less than the configuration with a larger d_l.
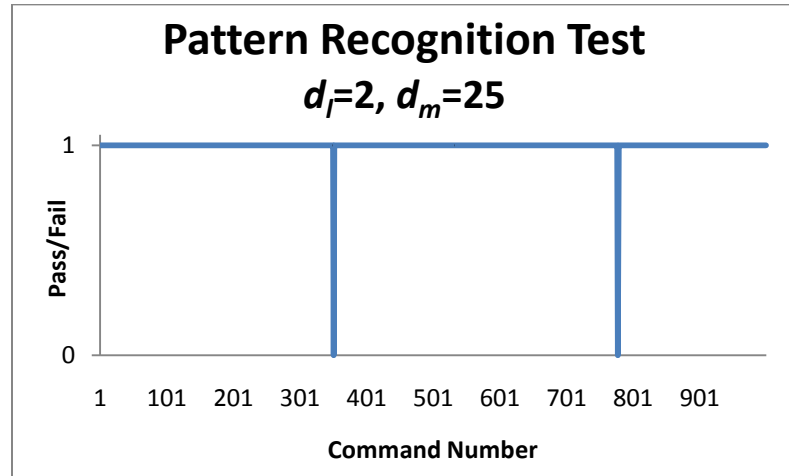

Figure 41. Pattern recognition pass and failure graph with d_l = 2, d_m = 25.

This is a significant improvement over the course of the test as the UNDO user and UNDO delta sync issue does not appear. This configuration makes it substantially easier to guess the delta value, but having a $(1/2)^n$ probability of passing n commands by guessing may be secure enough for some applications.

9.2.4 Pattern Recognition and Encryption Commands Test

This series of tests was to determine the effects of using both pattern recognition and encryption together to validate UNDO user commands. This test was first conducted with the auto-verify integrity feature disabled, allowing UNDO to focus on only processing UNDO user commands. Figure 42 shows that sending 1,000 user commands with both mechanisms enabled caused the same delta timing sync problem.
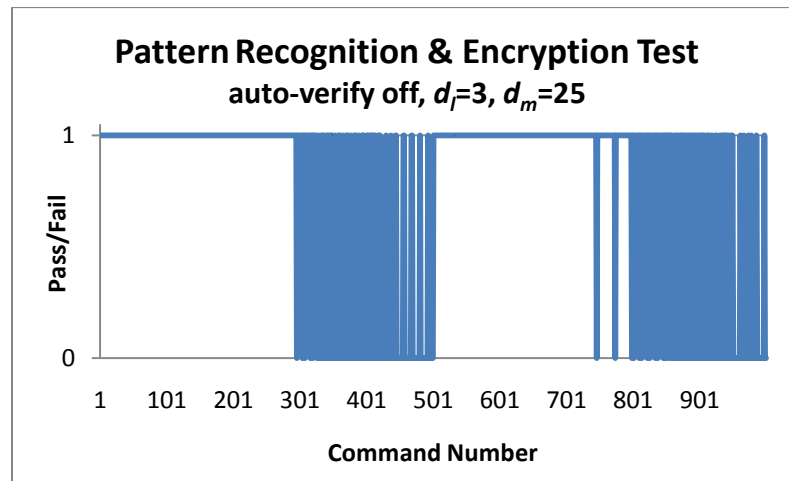
The test lasted 481,062 milliseconds.

Figure 42. Pattern recognition and encryption test results with auto-verify off.

The same test was then conducted with the auto-verify feature enabled. This test

concluded in less time than the first: 424,609 milliseconds. This shows that the sporadic

arrival times of the commands conforming to pattern recognition causes enough of a

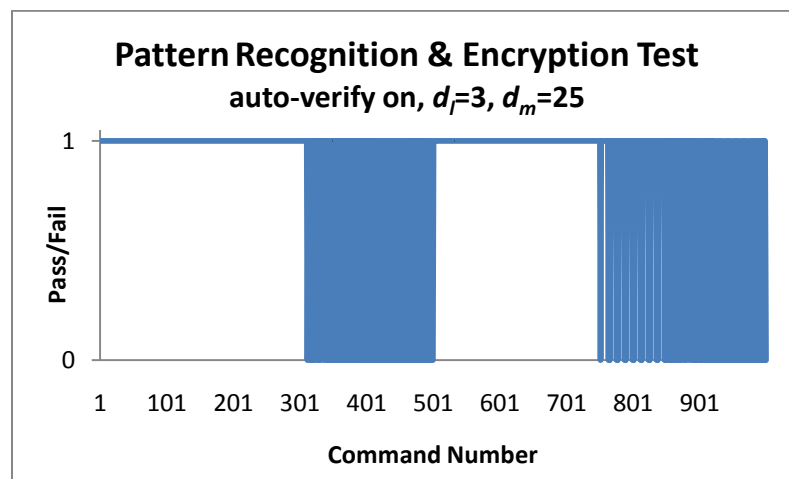delay to not stretch UNDO computing resources. Figure 43 displays the results.

Figure 43. Pattern recognition and encryption test results with auto-verify on.

9.2.5 Attack on Encryption

The security of the encryption was tested by sending 1,000 commands by an

UNDO user that did not have the correct secret key. The secret keys used for each of

these commands failed resulting in a 0% attack success rate, as shown in Figure 44. This
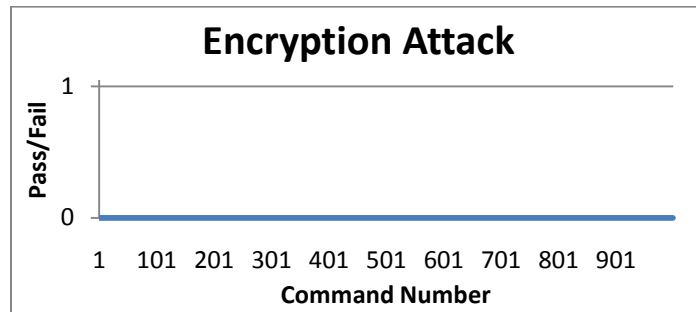
attack needed 221,515 milliseconds to complete.

Figure 44. Encryption attack success rate.

9.2.6 Attacks on Pattern Recognition

An attacker has a chance of defeating the pattern recognition scheme by

guessing the right delta at the right time. In a situation where the d_l was 3 and the d_m
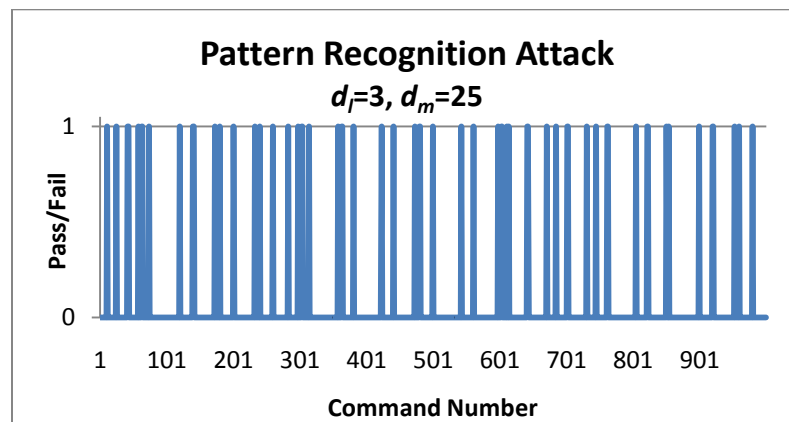
was 25, the attacker was successful in 60/1000 attempts.

Figure 45. Pattern recognition attack with d_l = 3 and d_m = 25.

In a situation where the d_l was 3, but the d_m was only 15, the attacker was successful in 30/1000 attempts.
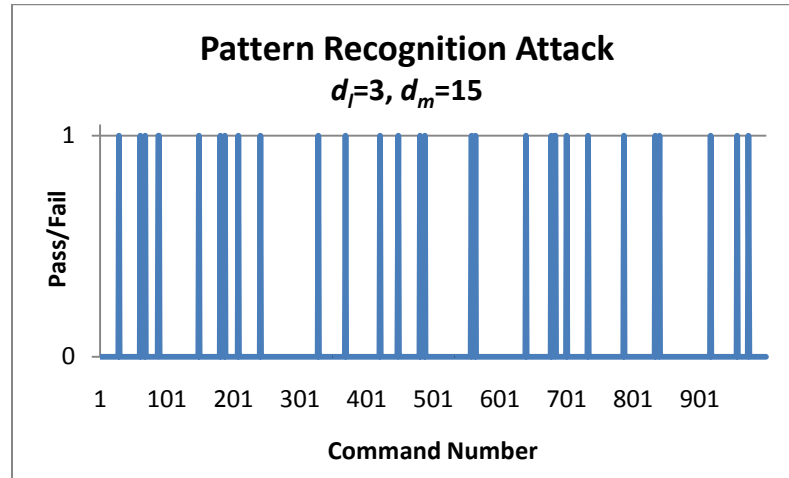

Figure 46. Pattern recognition attack with d_l = 3 and d_m = 15.

Using a smaller d_l was shown to greatly increase the attacker probability. In a situation where the d_l is 2 and the d_m is 25, the attacker actually has a 50% chance of guessing each delta correctly. Figure 47 shows that the attacker was close, passing 414 out of 1,000 commands.
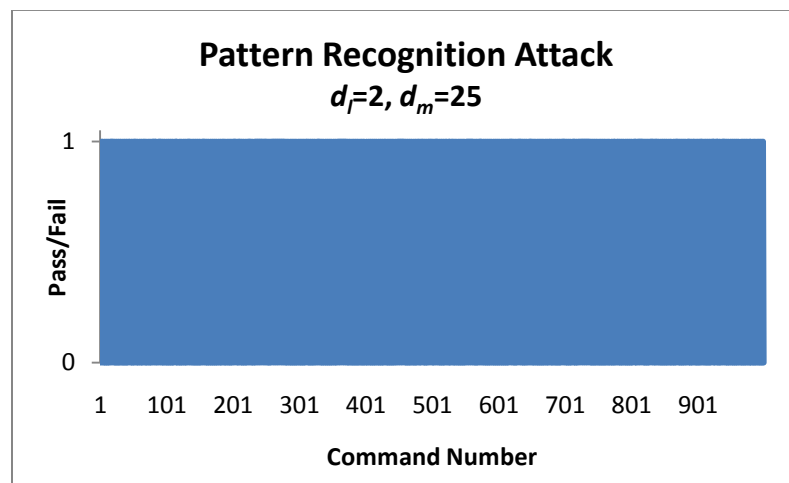

Figure 47. Pattern recognition attack with d_l = 2 and d_m = 25.

10. Conclusion

This thesis explored the need to protect digital data in various applications. Examples of successful attacks against seemingly secure targets were discussed to highlight this need. These examples also supported the assertion that data are difficult to protect. When focusing on the integrity of data, this thesis showed how the described TTP approach provided a solution to this problem under the appropriate circumstances. This approach focused on efficient recovery from compromise rather than traditional prevention means.

The UNDO system was developed as a functional representative of this security scheme. UNDO was able to appropriately monitor a file and database system based on administrator priority specifications. It was also able to process authorized user modifications to the monitored system. Testing was conducted with this implementation to prove its feasibility in a real-world application. Within the scope presented in this thesis, this approach was shown to have the power to negate the work of an attacker.

11. Future Work

The main purpose of this project was to provide a proof of concept, and a number of assumptions were made throughout the thesis which narrowed the scope of the project. As a result, this system, as described, is not yet ready to be a marketable product. The gap between this project's current status and a market-ready product allows for future development efforts.

A fundamental limitation of the proposed system was its inability to handle an attack beyond the scope of a nuisance-type attack. This limitation was mainly due to the possibility of a concerted, distributed, attack. If the system was set to refresh every fifteen minutes, an attack can be timed to hit every sixteenth minute, causing a down time of fourteen out of fifteen minutes. An analysis of this type of threat and appropriate mitigation for it would increase the applicability of this solution to a wider range.

System effectiveness is dependent on timely verification of data, and a known efficient and cryptographically secure algorithm was employed. However, this hierarchical hash generation and verification approach was not optimized and can be improved. Using the documented performance of this algorithm and knowing that UNDO will be verifying numerous hashes will allow one to develop a more applied approach. The act of designing a cryptographic algorithm, analyzing its efficiency, and proving it is secure is a project in itself. A future researcher can refine this

approach by designing and analyzing such an algorithm.

References

[1] E. Skoudis and T. Liston, *Computer Hack Reloaded: A step-by-step guide to computer attacks and effective defenses*. Upper Saddle River, N: Prentice Hall, 2006.

[2] M. Stamp, *Information Security: Principles and practice*. Hokoken, New Jersey: John Wiley & Sons, 2006, pp. 54-55.

[3] B. Schneier, "Cryptographic Design Vulnerabilities," *IEEE*, vol. 31, no. 9, pp. 29-33, Sept. 1998.

[4] K. McCarthy, "Hack Your Bank for $995: Cambridge Boffin Post Gory Details on Web," *The Register*, Nov. 9, 2001. [Online]. Available: www.theregister.co.uk/2001/11/09/hack_your_bank. [Accessed Apr. 12, 2009].

[5] J. Leyden, "Brit Charged with Hacking Pentagon, NASA: Military Left Windows Server Wide Open to Attack," *The Register,* Nov. 13, 2002. [Online]. Available: http://www.theregister.co.uk/2002/11/13/brit_charged_with_hacking_pentagon. [Accessed: Apr. 12, 2009].

[6] D. Ferbrache, "The Nature of the Hacking Threat to Open Networks," *Information Security – Is IT Safe?*, IEEE Colloquium, no. 1996/151, pp. 7/1-7/3, June 1996.

[7] Ionx, "Data Sentinel File Monitoring System," [Online]. Available: http://www.ionx.co.uk/html/products/data_sentinel. [Accessed Apr. 4, 2009].

[8] Runtimewear, "Sentinel," [Online]. Available: http://www.runtimeware.com/sentinel.html. [Accessed Apr. 4, 2009]

[9] R. Rivest, "RFC1321: The MD5 Message-Digest Algorithm," *IETF*, Apr. 1992. [Online]. Available: http://www.faqs.org/frcs/frc1321.html. [Accessed Feb. 7, 2009].

[10] J. Katz, Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton, FL: Chapman & Hall/CRC, 2008.

[11] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking ciphers in the real world*. Hoboken, NJ: John Wiley & Sons, 2007.

[12] H. Garcia-Molina, J. D. Ullman, J. Widom, *Database Systems: The complete book*. Upper Saddle River, NJ: Pearson Prentice Hall, 2009.

[13] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental Cryptography: The Case of Hashing and Signing," *Advances in Cryptography – Crypto 94 Proceedings, Lecture Notes in Computer Science*, vol. 839, Springer-Verlag, 1994. [Online]. Available: http://cseweb.ucsd.edu/~mihir/papers/inc1.pdf. [Accessed: Sept. 19, 2009].

Appendix A: UNDO Screenshots

Figure 48. Screenshot of UNDO prior to administrator authentication.

Prior to authentication, all functionality is disabled. The authentication screen is accessed by selecting "Authenticate" under the File menu.

Figure 49. Screenshot of UNDO administrator authentication box.

The UNDO administrator presents a username and password which is also used to establish a connection with the database.
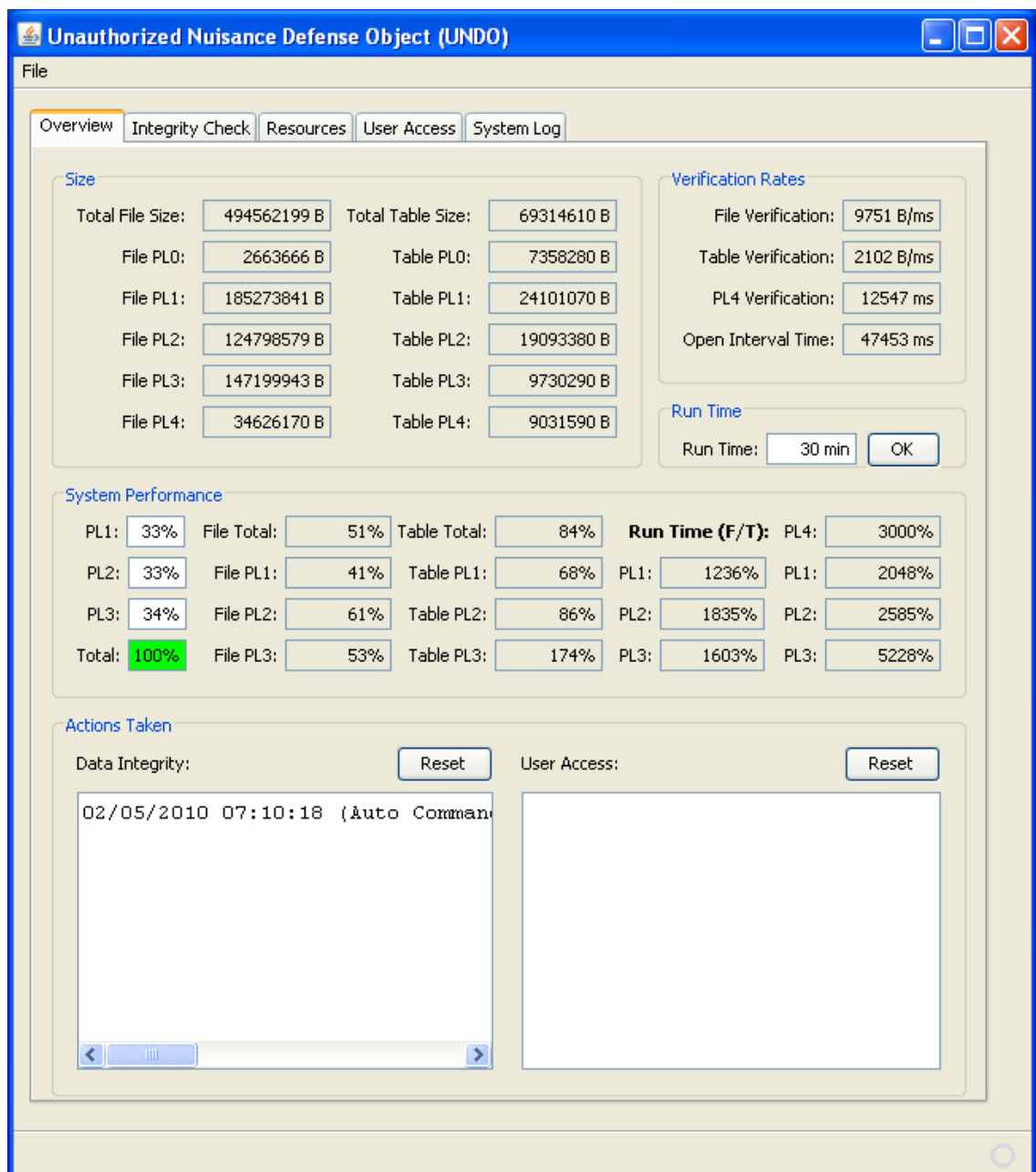
Figure 50. Screenshot of UNDO overview tab.

The overview tab displays metrics and UNDO system actions. .

Figure 51. Screenshot of UNDO integrity check tab.

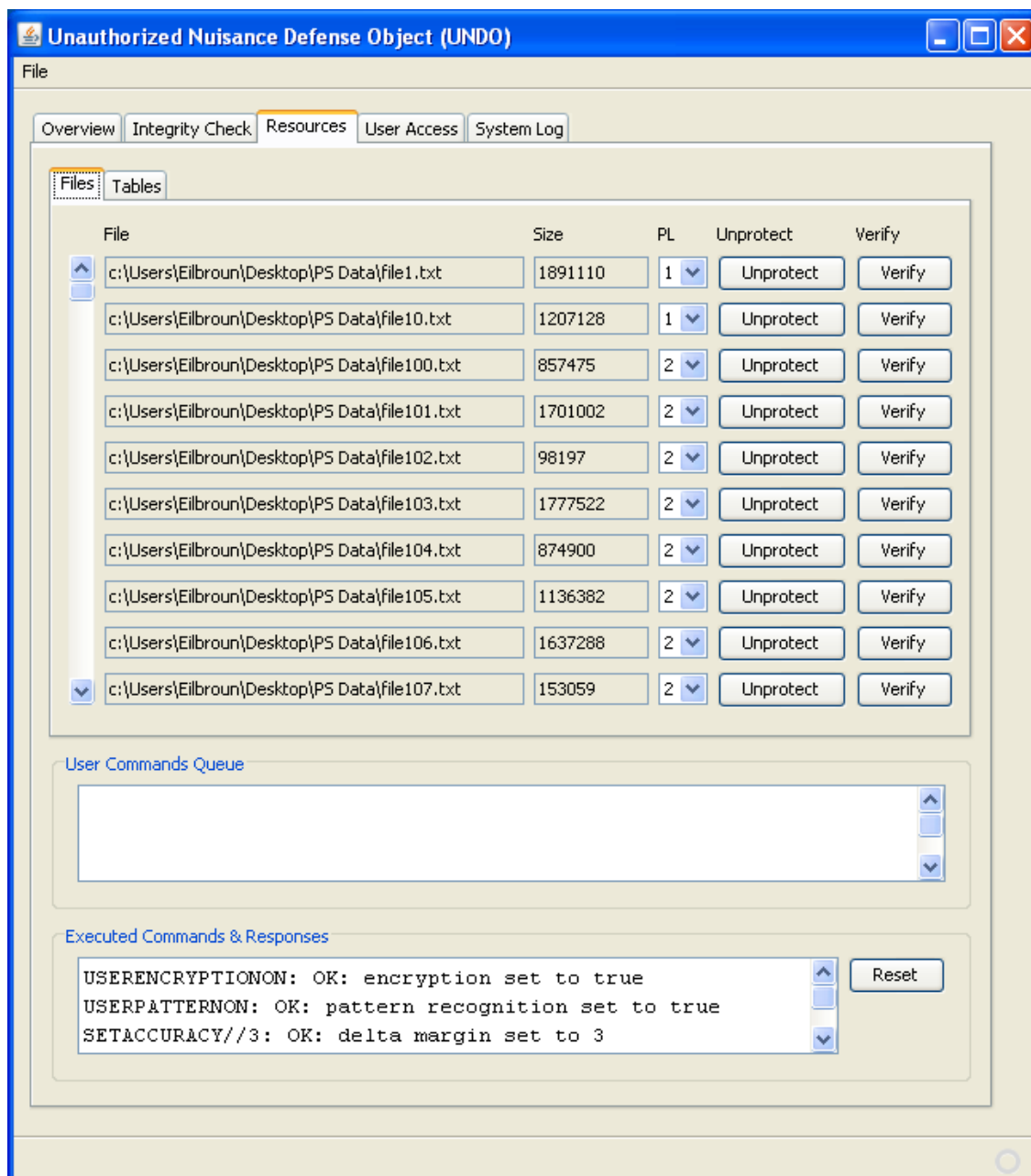The integrity check tab allows configuration of integrity verification and new protected resource selection.

Figure 52. Screenshot of UNDO files tab selected under resources tab.

The resources tab with the files tab selected allows an administrator to manage protected files.
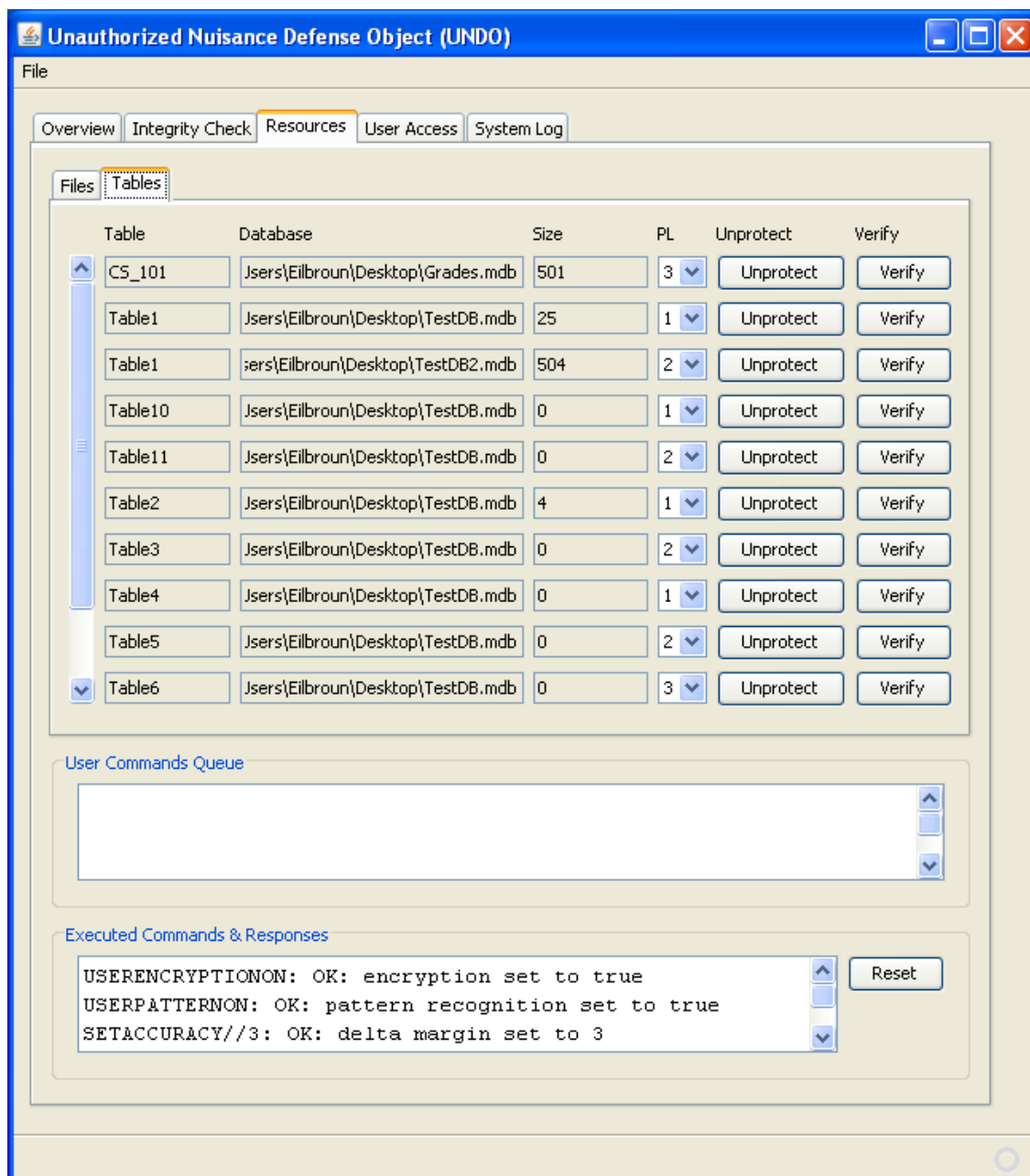
Figure 53. Screenshot of UNDO tables tab selected under resources tab.

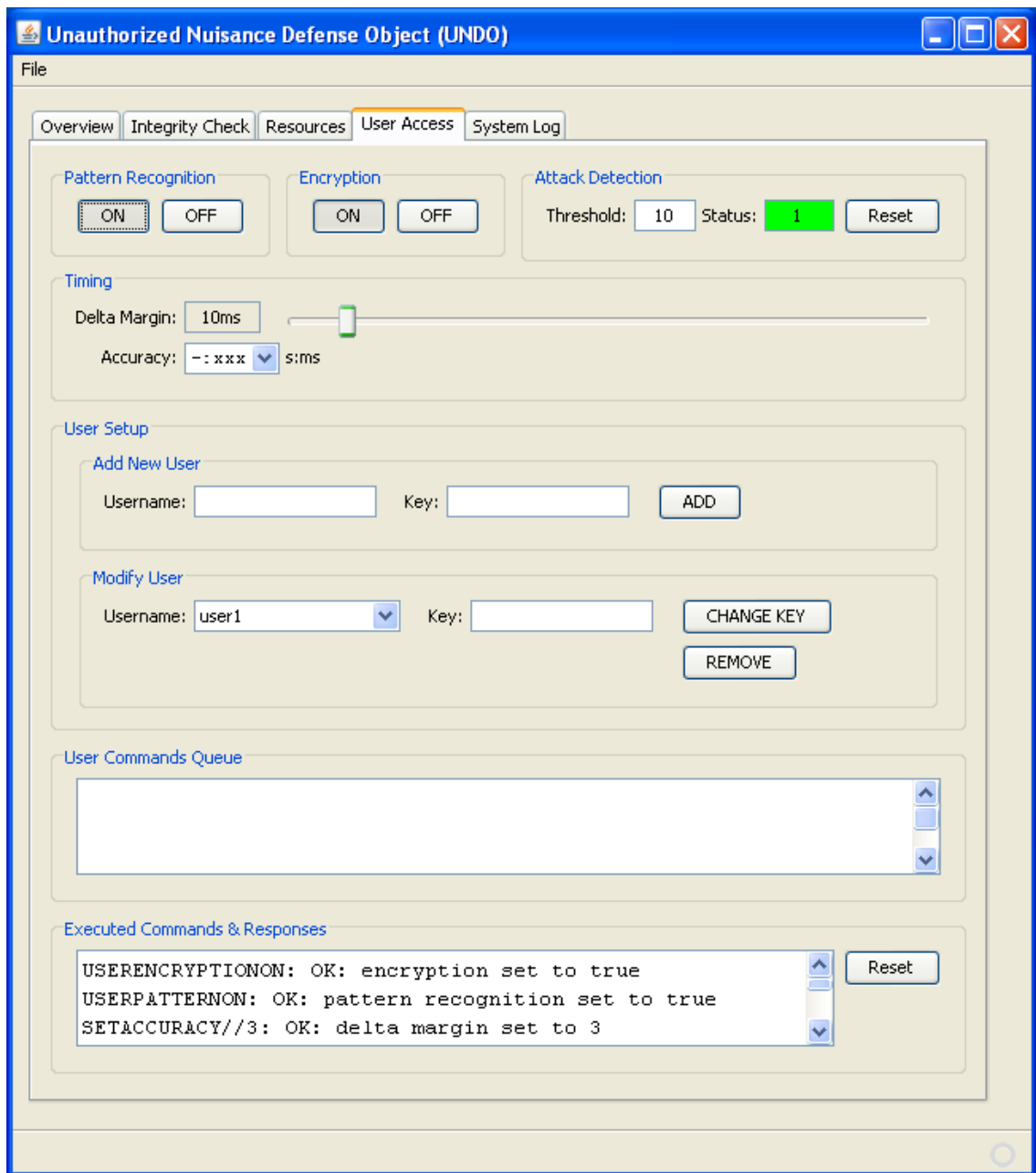The resources tab with the tables tab selected allows an administrator to manage protected tables.

Figure 54. Screenshot of UNDO user access tab.

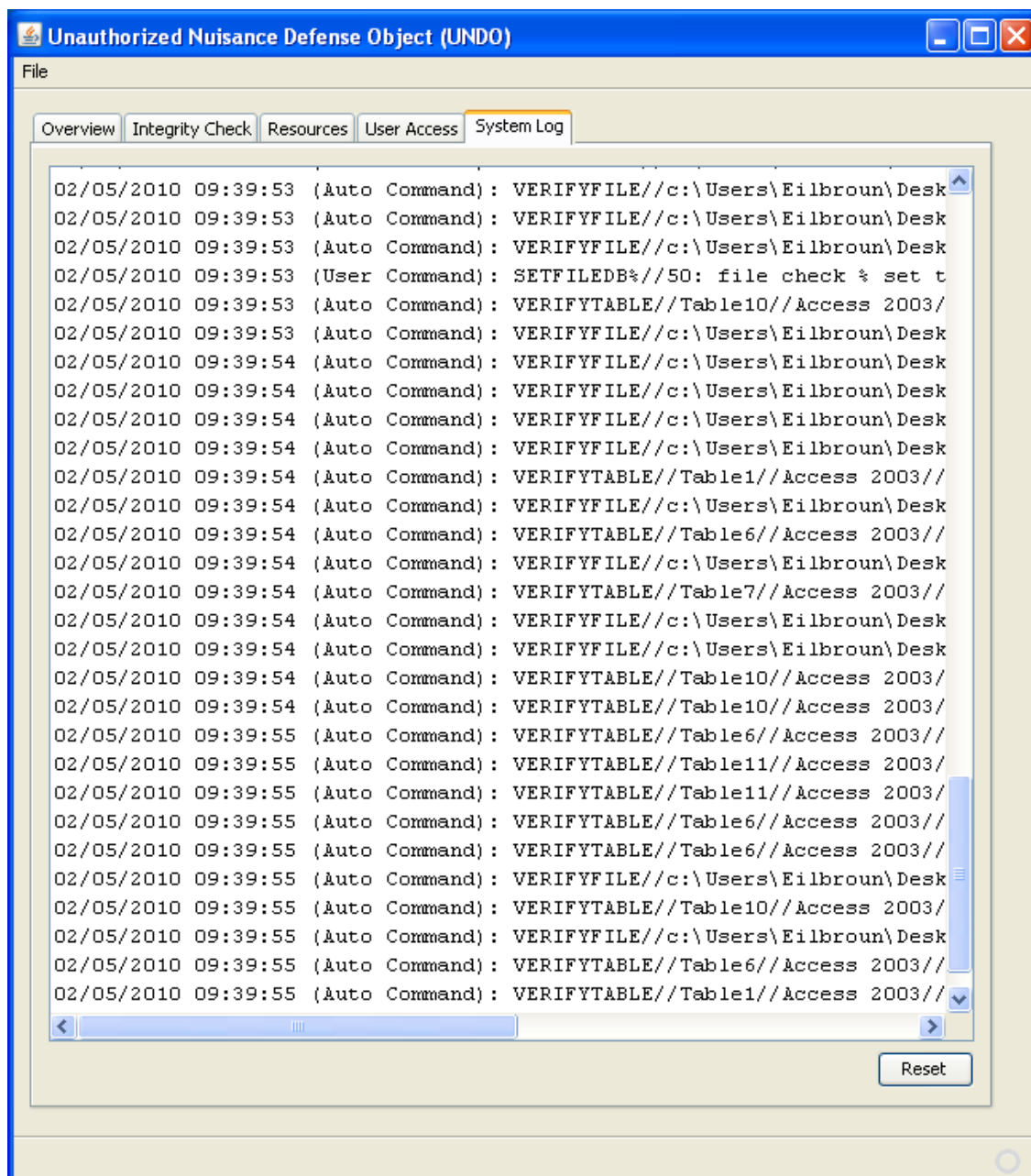The user access tab is used to configure UNDO user modification commands.

Figure 55. Screenshot of UNDO system log tab.

The system log displays all undo activity including simple verifications of protected resources.