2013

Detection and Identification of Software Encryption Solutions in NT-based Microsoft Windows Operating Systems

Julian Breyer
West Virginia University

Detection and Identification of Software Encryption Solutions in NT-based Microsoft Windows Operating Systems

Julian Breyer

Thesis submitted to the
Benjamin Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Dr. Roy Nutter, Ph.D., Chair
Dr. James Mooney, Ph.D.
Dr. Katerina Goseva-Popstojanova, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2013

Keywords: Encryption; Hard Drive Encryption; Computer Forensics; Live Forensic Analysis

ABSTRACT

Detection and Identification of Software Encryption Solutions in NT-based Microsoft Windows Operating Systems

Julian Breyer

As encrypted information is very difficult or impossible to reconstruct, there are many situations in which it is critical to detect the presence of encryption software before a computer is shut down. Currently there is no solution that reliably identifies installed encryption software.

For this investigation, thirty encryption software products for Microsoft Windows based on the NT-kernel have been identified and investigated. Operating system dependent factors such as registry, file attributes, operating system attributes, process list analysis and independent factors such as file headers, keyword search, Master Boot Record analysis as well as hashing of software components were investigated and allow the identification of these programs. The most reliable detection rate is achieved through a combination of the aforementioned factors.

To my family

Acknowledgements

I would like to thank my advisor, Dr. Roy S. Nutter, for his encouragement and assistance during my time as a graduate student as well as for many valuable insights into the forensic process during the many courses I took with him.

I also owe many thanks to the other two members of my thesis committee, Dr. James Mooney and Dr. Katerina Goseva-Popstojanova, for the great instruction in their respective fields of interest and for kindly agreeing to serve on my thesis committee.

My research was partially funded by the National White Collar Crime Center (NW3C) as part of their mission to provide free high-quality tools and training to law enforcement and I am grateful for the opportunity their funding provided me with.

Last but not least, I would like to thank my lab partners, Sean Adam and Neil Bowman for their open ears and invaluable advice.

Table of Contents

Table of Figures

Index of Tables

Chapter 1 - Introduction

The most common computer forensics analysis technique is known as “post-mortem analysis” or “dead analysis”. Responders disconnect any computer they suspect may contain evidence from its power source to prevent any accidental or malicious disposal or alteration of evidence and subsequently remove the disk drive from the computer [1]. The disk is then examined on a “trusted” computer using one of the many available computer forensic tools, such as EnCase, FTK or Autopsy [2]. The intent behind this “dead analysis” process is to examine the contents of the suspect’s hard disk using a write blocker and another computer. This eliminates the risk that the “trusted” computer used for forensic analysis will write anything to the disk under examination [3].

While this method has proven value for digital forensics, it also has a caveat: if the contents of a hard drive, or a portion thereof, were encrypted prior to the original computer’s shutdown, the contents of that hard drive may be irretrievably lost [4].

Live forensic analysis, on the other hand, is performed directly on the suspect’s computer. The computer is not shut down and volatile evidence, such as memory content, that is lost during shutdown is preserved. While this method allows investigators to examine the computer in the exact state in which it was seized, it also bears the risk that volatile and stored evidence might be changed or destroyed. This can be the result of faulty software used for the investigation or of a deliberate attempt to hide or destroy evidence.

Currently there is no comprehensive solution available that reliably detects a variety of encryption software products and can be adapted to discover new or changed software products. This investigation identifies and examines a number of operating system dependent and independent factors by which known Windows encryption packages can be detected and

identified and which can be employed in a signature-based approach. A framework termed ALERT (Automated Live Encryption Recognition Tool) has been developed as a proof of concept that encryption software can be reliably detected with minimal human intervention during a live forensic investigation. To examine the reliability and effectiveness of each factor by itself and of combinations of factors in unison, 25 experiments have been conducted and analyzed.

Thesis Statement

This investigation examines if all of the thirty most common software encryption solutions identified for NT-based versions of Microsoft Windows are reliably identifiable through the live analysis of a combination of operating system dependent and independent factors. It furthermore analyzes whether one or more combinations of factors exists that is more reliable in terms of precision, recall and accuracy than any single factors in isolation.

1.1 Predictions

1.1.1 Best Performance When a Signature Exists

It is expected that the Operating System dependent factors will produce the best results in terms of precision for cases in which a signature is available for the particular version to be detected. File Header analysis is expected to perform as well or better than any other factor in terms of recall but at the expense of execution speed.

1.1.2 Best Performance When no Signature Exists

Overall, it is expected that the keyword search will perform as well or better than any other factor in terms of recall when no signature exists for the software product. For Full Disk Encryption programs, the heuristic approach is expected to yield the best results in terms of recall while the Operating System dependent measures should yield very low recall but high precision values.

1.1.3 Best Overall Performance

It is expected that there exists a combination of factors C such that for any factor F in isolation $recall_C >= recall_F$ because encryption software, like viruses, attempt to hide their presence by using leaving as few detectable traces in the Operating System as possible.

Chapter 2 - Background

2.1 Encryption

2.1.1 Historical Background

The protection of sensitive information has been important to societies since the inception of written communication. One of the first documented ciphers is the Caesar Cipher used by Julius Caesar to protect his correspondence from the prying eyes of his enemies. In this simple encryption system, letters of the alphabet are shifted, or transposed, by a certain number of steps. The sender and receiver might, for example, agree that all letters are transposed by three. As illustrated in Figure 1, a “B” would become an “E”, a “C” would become and “F” and so forth. The word “FILE” would become “ILOG”. What has not changed is the length of the “encrypted” word. Additionally, if one had access to a large enough sample of the encrypted text and knew which language it was written in and with which frequency certain letters occur in that language, it would be a trivial task to decipher the message.

While this might seem insecure today, the fact that few people could read even unencrypted communications at the time this cipher was conceived provided a certain amount of security. Today, this type of cipher is generally known as a “shift cipher” or “mono-alphabetic substitution” [5].

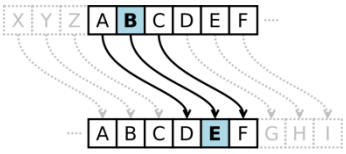
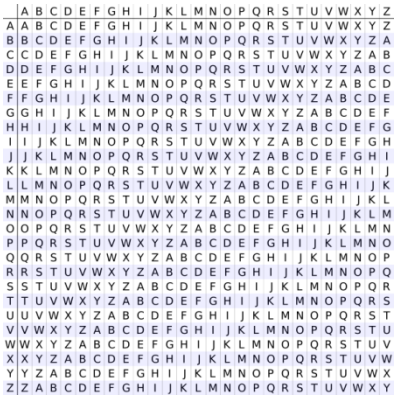


Figure 1 - Shift Cipher (Source: Wikipedia)

As time progressed, new and more secure ciphers, as well as ways to break them, were devised. It was not until World War II, however, that computers were first used to attack cipher text. The Nazis had devised an encryption machine (the Enigma machine) to safeguard their

correspondence. Unlike the early shift ciphers, the Enigma relied on a more intricate form of cryptography known as “Poly-alphabetic Substitution” that had been developed in the 15th century. Poly-alphabetic substitution ciphers rely, as the name suggests, on multiple different alphabets and a key. The substitution is based on the alphabet belonging to a particular part of the key, the numerical value assigned to a letter (e.g. 1 for an “a”) and the corresponding substitution [5].



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 2 - Vigenère Table used for poly-alphabetic substitution. (Source: Wikipedia)

Since messages were commonly transmitted via wire or radio, the Allied Forces were able to intercept great amounts of cipher text. Dr. Alan Turing, a well-known English mathematician then working at a military facility in Bletchley Park, England, devised a method to use mechanical computers to decrypt the intercepted messages as part of a project named “Ultra”. While this was laborious and often did not lead to the successful decryption of a message, it built the foundation of much of today’s cryptanalysis [6].

In the light of the Cold War, the necessity to secure national secrets against opposing forces eventually led to the development of a National Institute of Standards and Technology (NIST) endorsed encryption standard now known as Data Encryption Standard (DES) [7]. While the standard was regarded as secure at the time, it used a secret key that had to be shared with others authorized to access data encrypted in this manner. Additionally, as shown by Diffie and

Hellman, its key length of 56 bits was too short to keep information secure for long when the computers used to attack the data were becoming more powerful every year [5].

The widespread access to computer networks and networks of networks, such as the internet, eventually led to a call for a new and improved standard. This occurred with the replacement of DES by the Advanced Encryption Standard (AES) in 2001 [8]. At the same time other mechanisms for communications, such as Secure Socket Layer (SSL) / Transport Layer Security (TLS) encryption were designed. New encryption algorithms using massively long encryption keys that are based on the complexity of factoring relatively prime numbers as well as the advent of asymmetric encryption have made today's encryption easy to use, virtually transparent and ubiquitous.

Most of today's operating systems are equipped with powerful encryption software that can be used to encrypt single files or an entire file system and require virtually no technical knowledge. Additionally, several third party solutions are available to both private individuals and the industrial world [5]. In addition to software-based encryption, many manufacturers of storage and computing equipment offer hardware implementations of current encryption algorithms that are faster and more difficult to circumvent than their software equivalents.

2.1.2 Encryption Algorithms Commonly Used in Storage Encryption

2.1.2.1 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) was first published in 1998 as "Rijndael" by Rijmen and Daemen and later standardized in 2001 as a replacement for the aging Data Encryption Standard (DES). The algorithm employs a Substitution-Permutation network with a block size of 128 bit and a key length of 128, 192 or 256 bit. It further uses a symmetric key; i.e.

the same key is used for encryption and decryption. AES is currently approved by the National Security Agency (NSA) for information classified as Top Secret [9].

A Substitution-Permutation network applies a number of logic or algebraic operations to transform the input data. These operations are then repeated a number of times. The number of cycles depends on the length of the key. In AES, 10 cycles are used for 128 bit keys, 12 cycles for 192 bit keys and 14 cycles for 256 bit keys. Each cycle follows the following format:

1. **Key Expansion**; round keys are derived using Rijndael's key schedule.
2. **Initial Round**

   - AddRoundKey: Bitwise operation that performs $[round\ key] \oplus [input]$.
3. **Rounds**

   1. SubBytes: Each byte of input is replaced with another one according to a lookup table.

   2. ShiftRows: Each row of the block is shifted cyclically.

   3. MixColumns: Exchanges the order of the columns of the block cyclically.

   4. AddRoundKey
4. **Final Round**

   1. SubBytes

   2. ShiftRows

   3. AddRoundKey

2.1.2.2 Serpent

The Serpent algorithm, proposed in 1998 by Anderson, Biham and Knudsen, is a 32-round Substitution-Permutation network that utilizes a 128 bit block size and a key length of 128, 192 and 256 bit. Like Rijndael, Serpent was a finalist for the AES standard. All steps of the

algorithm can be performed in parallel, which improves its performance but also renders it more susceptible to cryptanalysis [10].

The algorithm consists of

1. An initial permutation
2. 32 rounds of
   - key mixing
   - a pass through S-Boxes (substitution boxes)
   - a linear transformation in all but the last round
3. A final permutation

2.1.2.3 Twofish

Twofish was proposed by Schneier, Wagner, Hall, Kelsey, Whiting and Ferguson in 1998 – also in competition for the AES standard. Just as Rijndael and Serpent, it uses a block length of 128 bits and a key length of 128, 192 or 256 bits but unlike the other two, it is implemented as a Feistel Network [11].

The algorithm consists of

1. An initial input whitening
2. 16 rounds of
   - 4 S-Boxes
   - maximum distance separable (MDS) mapping
   - Pseudo-Hadamard transforms (PHT)
   - a bitwise shift
   - multiple XOR-operations
3. Undo of the last swap operation

4. Output whitening

2.1.3 Complexity of Breaking Current Encryption Standards

The difficulty of breaking modern day encryption is best illustrated with a case investigated by the Federal Bureau of Investigation (FBI) and the Brazilian National Institute of Criminology in 2009. While investigating a Brazilian banker charged with money laundering, the authorities seized five hard drives that all exhibited military-grade AES 256-Bit encryption through a common Full Disk Encryption tool called TrueCrypt [12]. Since in Brazil as well as the United States, a suspect cannot be compelled to surrender an encryption key or password, the FBI attempted to decrypt the contents of the hard drive using various brute-force methods such as dictionary attacks. Twelve months later, the agents gave up having made no progress and returned the hard drive to the Brazilian authorities [13].

All of the algorithms described in this section were deemed sufficiently secure by NIST to be accepted as finalists for the AES standard. While Rijndael ultimately won the contest, all three algorithms are thought to be very resilient against short-cut attacks, leaving brute force attacks (such as dictionary attacks) as the only viable option to decrypt the hard drive.

Most hard- and software implementations today recommend a key length of 256 bits for confidential information. Breaking a key of this length by brute force would require approximately 2^{255} operations. Given 50 of the most powerful super-computer available in 2012 (the 20-peta-FLOP Cray Titan at Oak Ridge National Labs) these machines could process a combined 10^{18} operations per second. At that rate, all 50 super-computers would require $1.86*10^{51}$ years to attempt every possible combination. For AES, it has been shown that the complexity for a 256-bit key can theoretically be reduced to 2^{176} operations, reducing the time to

$3*10^{27}$ years which is still far beyond feasible. The longest key successfully broken by this method had a length of 60 bits.

2.1.4 Detecting and Identifying Encryption

A method commonly employed in cryptanalysis is the analysis of the frequency with which symbols and characters occur in cipher text. If a simple cipher is employed, the frequency with which certain letters occur might provide useful information regarding the language of the clear text as well as a possible starting point for the decryption of the message. To avoid this, strong encryption methods strive for a roughly uniform distribution of all symbols so that no conclusion about the encryption system, the plain text message or even just the presence of encryption can be drawn. This makes the detection and identification of a particular encryption algorithm very difficult if not impossible.

2.1.5 Prior Attempts to Detect Encrypted Data Storage

Due to the inherent difficulty of detecting and classifying the presence of encryption itself, a number of companies have developed software tools to detect a small subset of commonly used encryption programs through alternate means, such as a search for a particular dynamically linked library (DLL) known to be used by the programs. JADsoftware's "Encrypted Disk Detector" (EDD) [14] is capable of detecting TrueCrypt, PGP and BitLocker through analysis of the Master Boot Record (MBR). TechPathWay's "ProDiscover" [4] does not detect any particular encryption suite but presents a user with the contents of the Master Boot Record (MBR). LEAP, a live encryption analysis program developed by another Master's candidate at West Virginia University, can detect a number of encryption programs, including Microsoft BitLocker, but relies mainly on the analysis of file names in particular locations on the hard drive. Furthermore, there are several websites that claim to offer software that can classify the

algorithm used to encrypt a hard drive from patterns formed by the use of different algorithms but within the forensics community, these are widely regarded as a fraud. Other solutions may exist but were unavailable for analysis as they are often proprietary and classified and no information about their existence or functionality is made available to the general public.

2.2 Electronic Evidence and the Forensic Process

2.2.1 The Latency of Electronic Evidence

Digital computers are state machines. This means that at any given moment, a computer has a well-defined state that constantly changes based on the previous state, inputs and outputs and operations that need to be performed on those data. This state can contain valuable information about the prior and present use of the computer but it is also constantly under threat of change and therefore must be preserved to the greatest extent possible given the time available to an investigator to prevent information from being destroyed. In less abstract terms, information about programs and data currently executing on the computer as well as time stamps and administrative data can change at any given moment and may make the recovery of previous information difficult or impossible.

Some of the computer's data resides on long-term / semi-permanent storage usually referred to as "hard-drives", while other information is stored in the computer's volatile registers or its Random Access Memory (RAM). This type of memory relies on the presence of an electric current to preserve the information stored within it. Once this current is lost (e.g. due to a loss of power or controlled shutdown), the information quickly dissipates and becomes irretrievable. Additionally, magnetic storage can easily be altered or destroyed if the storage medium is handled improperly. A drop and even moderate impact can damage the fragile magnetic platter

on which information is stored. Electromagnetic fields can also alter the data stored on the magnetic disk [15].

This inherent volatility of electronic evidence, just as the volatility of fingerprints or other trace evidence requires that the evidence be collected, handled, examined and preserved with the utmost care and in a way that holds up in a court of law. This requires training and reliable and well-tested tools and processes.

2.2.2 The Classical 4-phase Approach

To accommodate the stringent requirements for the permissibility of electronic evidence at trial, digital forensics typically employs a four-phase approach. It consists of collection, examination, analysis and reporting phases.

2.2.2.1 The Collection Phase

Typically, the collection phase takes place at the site of the incident. Investigators first document the site. This may include accounts of what technology was found at the scene and what the devices looked like or displayed when they were found. This account is usually augmented by photographs and possibly screens shots. Once the original state of the site has been documented, the incident responder will either conduct a live analysis of the device or disconnect it from its power source. The live analysis will be discussed in a later section of this chapter. Once the computer has been separated from its power supply, all incoming and outgoing connections to peripherals are labeled and documented [16]. It has become increasingly difficult to identify and recognize all relevant devices since many of them look inconspicuous or require training to be recognized. The computerization and networking of seemingly benign items (cars,

phones, copy machines, coffee makers etc.) that may contain important evidence further complicates this task.

2.2.2.2 The Examination Phase

The examination phase concentrates on the extraction of evidence from the seized devices. This includes the detection and identification of the evidence as well as the explanation of its origin and significance [15]. The work in this phase is commonly conducted by forensic examiners with the help of well-tested software suites such as Guidance Software's "EnCase" [2]. Since in a computer most evidence such as artifacts of a user's visited websites or a timeline of tasks conducted on the computer in a 24 hour window cannot be seen with the bare eye, it must be made visible by the forensic examiner so that its significance can later be determined by a criminal investigator.

The volatile nature of the evidence again becomes important. If a tool or method used by the examiner were to alter, erase or add information to the secured evidence, the evidence would be useless in court. Therefore, all tools, physical or software, must be subjected to thorough testing to demonstrate that they do not alter the evidence in any way. Likewise, the examiner must have a thorough understanding of the function of a computer and how the evidence might have been produced as it would otherwise be easy for an investigator to interpret the results of the examination incorrectly and mistake benign data for incriminating evidence or vice-versa.

Examination is commonly conducted in one of two different ways: as "live" analysis on a running system or as "dead" or "post-mortem" analysis of the secured storage device via a second, trusted set of hardware and software. "Post-mortem" analysis should generally be regarded as favorable as the use of a separate trusted system offers more safeguards against the inadvertent alteration of evidence. Furthermore, the fact that the computer under investigation is

shut down guarantees that no program on it can be executed that would hide, alter or dispose of incriminating evidence.

As Brian Carrier notes [1], there is virtually nothing that "live" analysis can accomplish that cannot also be accomplished via "post-mortem" analysis. This, however, is only true with certain qualifications. If the computer system is an essential production system, the cost of removing and replacing it might be prohibitive. Furthermore, if the contents of the computer system's hard drive are encrypted, "post-mortem" analysis of its stored contents is only possible if the encryption key or password can be obtained. Without this, the information stored on the computer becomes inaccessible as soon as the computer is shutdown or its operating system is locked due to inactivity.

2.2.2.3 The Analysis Phase

The analysis phase is conducted by the criminal investigator and interprets the findings of the examination phase. While the examiner and investigator may be the same person, the two phases serve essentially different purposes. The examination phase concentrates on the technical abstraction of evidence from the seized computer(s) while the analysis phase attempts to relate the evidence secured from the computer to the crime under investigation.

2.2.2.4 The Reporting Phase

The reporting phase creates a seamless report of all of the actions performed during the previous three phases. During discovery or at trial, forensic examiners should not only present the results of their examination but must also convince all parties involved that their method was sound and that they were qualified to conduct the analysis.

2.2.3 Order of Volatility

The Internet Engineering Task Force (IETF) has published in RFC 3227 the order in which evidence decays and therefore should be secured [17]:

- registers, cache
- routing table, arp cache, process table, kernel statistics, memory
- temporary file systems
- disk
- remote logging and monitoring data that is relevant to the system in question
- physical configuration, network topology
- archival media

2.3 Files and File Systems

2.3.1 Files

A file is, in its simplest form, a logical collection of characters of which a physical representation is stored in volatile or non-volatile memory. In ASCII, one of the most common character sets in use today, each character is represented by seven bits for a total of 128 different representable characters. On magnetic devices, each bit comprising a character is simply described as being represented by the polarity of a particular storage cell on the magnetic platter. In volatile and solid state memory, it is often represented as the presence or absence of a charge in a storage cell.

2.3.2 File Systems

Files are not usually stored in coherent units on the storage device but are often spread out across a storage device for efficiency reasons.

In order to manage the physical space available and to locate and retrieve the various pieces of a file, Operating Systems commonly employ file systems as another abstraction. A file system is a collection of files and a set of functions to store, modify, delete, identify and locate any particular file. Commonly, file systems also contain means to protect files and to manage and compact storage units and thus increase the efficiency, usability and security of the storage device [3].

2.3.3 File Attributes and Metadata

Files are often augmented by meta-information; i.e. information about the file. The most common metadata are the filename and the modified, accessed and created time stamps associated with each file. These are usually set by the file system but may be altered manually. Other file attributes can express that a file is write-protected, hidden, encrypted or compressed. While all of these attributes or metadata are easy to falsify or alter, they can contain vital information about the file [3].

2.4 Measures of Reliability

2.4.1 Confusion Matrix

A confusion matrix or confusion table is an illustration of the performance of an algorithm typically employed in artificial intelligence and machine learning. Each column represents an expectation while each row represents actual results. These results can be grouped into four categories: true positives, false positives, true negatives and false negatives. Presented with an algorithm that is supposed to classify cats in a group of cats or dogs, true positives would be the instances in which a cat was correctly classified as a cat, true negatives would be cases in which a dog was not classified as a cat, false positives would be cases in which a dog was

classified as a cat and false negatives would be cases in which a cat was not classified as a cat. With regards to this investigation, true positives are considered cases in which encryption software was present and detected, false positives are cases in which encryption software was not present but is detected, true negatives are cases in which no encryption software was present and none was recognized and false negatives are cases in which encryption software was present but not detected.

Predicted / Actual	Encryption Software	No Encryption Software
Encryption Software	True Positive	False Negative
No Encryption Software	False Positive	True Negative

Table 1 - Confusion Table for Encryption Detection

2.4.2 Precision

Precision is the ratio of the number of true positives to the sum of the numbers of true and false positives. For the work at hand, precision plays a minor role as it is more important to identify all true positives and minimize the false negatives than it is to identify only the true positives and minimize the false positives.

$$prec = \frac{A}{A + B}$$

Equation 1 – Precision

A here denotes the number of True Positives and *B* the number of False Positives.

2.4.3 Recall

Recall measures the ratio of the number of true positives to the sum of the numbers of true positives and false negatives. This value is important for this investigation because it indicates the predictive value of each factor under investigation. The recall will be the primary quality measure for each factor.

$$pd = recall = \frac{A}{A + C}$$

Equation 2 – Recall

A here denotes the number of True Positives and *C* the number of False Negatives.

2.4.4 Accuracy

Accuracy expresses the ratio of the sum of the numbers of true positives and true negatives to the sum of the numbers of true negatives, true positives, false negatives and false positives. While it has no particular relevance to this investigation it will be computed for completeness.

$$accuracy = \frac{A + D}{A + B + C + D}$$

Equation 3 – Accuracy

A here denotes the number of True Positives, *B* the number of False Positives, *C* the number of False Negatives and *D* the number of True Negatives.

2.5 Software Detection

2.5.1 Signature Based Software Detection

In the context of virus and malware detection and identification, signatures are abstractions of one or more factors by which a particular program can be identified. To obtain these signatures, a piece of software is analyzed for common behavior or artifacts that are consistently created by the software and can that be detected. This may encompass file header values, code signatures, checksums or other criteria that can be contained in a file, the operating system, a directory, the boot sector, or any other detectable location.

This approach is very efficient in that multiple criteria can be combined into an abstraction that indicates the presence of a particular piece of software. Its major drawback is that only known programs can be identified this way. Even minor changes to the software may require the derivation of a new or alternate signature to guarantee that the alternate version is still detected.

2.5.2 Heuristic Software Detection

To mitigate the problems that may arise from outdated signature databases and yet unknown viruses, heuristic methods are often employed. A heuristic method does not detect any particular security threat but rather monitors the Operating System for suspicious behavior that is common for particular attacks. If, for example, a program requests to execute a certain sequence of suspicious commands or a kernel-level function that is normally reserved for the Operating System, the heuristic monitor can use this information to deduce the likelihood that the behavior is an indicator of malicious software.

2.5.3 The Balance between False Positives and False Negatives

Most consumer and enterprise grade software is expected to reduce the number of false positives to reduce user confusion and frustration. If a user is permanently alerted to security threats that turn out to be false positives, he will be more inclined to ignore all warnings. Thus, for this type of software it is preferable to miss a small percentage of possible threats in order to reduce the number of false positives.

For the detection of encryption software, the impact of false positives is assumed to be reversed: in order to minimize the risk of missing an encrypted volume even a considerable number of false positives is acceptable as long as the number of false negatives approaches zero. This is believed to be the case because the consequences of any false negative could be catastrophic and result in the loss of evidence. False positives, however, are negligible because computers are not routinely scanned for encryption software and thus the false positives do not amount to a persistent nuisance for the forensic analyst.

Chapter 3 - Experiments

3.1 Setup

3.1.1 Identified Encryption Software Packages

First, a list of common Full Disk Encryption (FDE) programs and container-based encryption software was compiled. This list encompasses a total of 30 programs and can be found in Table 2, section 3.1.2.

3.1.2 Derivation of Signature Values

In order to derive signatures for all of the thirty programs, a uniform approach was followed as shown below.

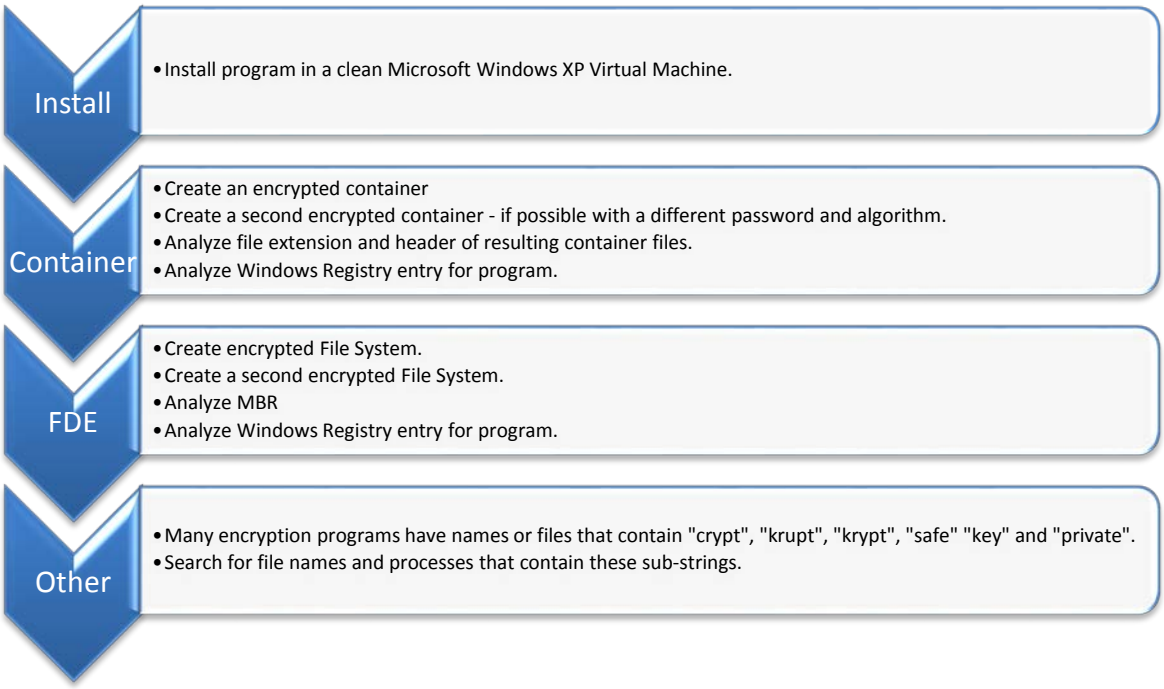


Figure 3 - Signature Derivation Process

Each of the thirty programs was installed in a clean Windows XP Virtual Machine (VM). Each program was then used to create an encrypted container (virtual drive) or file system. For

the full disk encryption programs, the Master Boot Record and Windows Registry were analyzed while for container based programs, the header and extension of the resulting encrypted container and Windows Registry were analyzed. Next, another container or File System was created (if possible, using a different encryption algorithm and password) and the aforementioned values were analyzed again for the new file container or File System.

If a factor remained constant between the two tests, it was recorded and the program was deemed identifiable by this particular factor. The two outliers are Microsoft's Encrypting File System, which can be identified only through a file attribute, and Microsoft BitLocker, which can be identified solely via an Operating System attribute.

Lastly, the entire File System and Process List were searched for file names and processes containing the sub-strings "crypt" although the list has since been expanded to also include the terms "krupt", "krypt", "safe", "key" and "private". Table 2 shows the results of this process.

Software	File Header	File Extension	File Attribute	Keyword Search	Master Boot Record	OS Attribute	Process List	Windows Registry
AdvancedFileSecurity	X	✓	X	✓	X	X		✓
ArchicryptLive5	✓	X	X	✓	X	X		✓
BestCrypt	✓	✓	X	✓	X	X		X
BestCrypt_FDE	X	X	X	✓	✓	X		X
BitLocker	X	X	X	X	X	✓		X
CompuSec	X	X	X	X	X	X		✓
CryptArchiveLite	✓	X	X	✓	X	X		✓
CypherixLE	✓	X	X	✓	X	X		✓
drivecrypt	X	X	X	✓	X	X		✓
E4M	X	X	X	X	X	X		✓
EFS	X	X	✓	X	X	X		✓
FreeOTFE	X	✓	X	X	X	X		✓
GilliSoftFDE	X	X	X	X	X	X		✓
LetEncrypt	X	X	X	✓	X	X		✓
Loop-aes	X	✓	X	X	X	X		X
Keyparc	X	X	X	X	X	X		✓
Kruptos2Professional	X	✓	X	✓	X	X		✓
MyWinLocker	✓	X	X	X	X	X		✓
n-CryptPro	X	X	X	✓	X	X		✓
NCrypt	X	X	X	✓	X	X		✓
PGPDisk	✓	✓	X	X	X	X		✓
PGP WDE	X	X	X	X	✓	X		✓
PrivateDisk	X	✓	X	✓	X	X		✓
PrivateSafeHD	X	X	X	✓	X	X		✓
R-Crypto	X	✓	X	✓	X	X		✓
SafeBoot	X	X	X	✓	✓	X		X
SafeticaPersonal	X	✓	X	✓	X	X		✓
SafeHousePro	X	✓	X	✓	X	X		✓
Sentry2020	✓	X	X	X	X	X		X
TrueCrypt	X	X	X	✓	✓	X		✓

Checkmarks (✓) denote that the software can be identified by the corresponding factor while X denotes that this is not the case.

Table 2 - Encryption Software and Identifiable Factors.

3.1.3 Heuristic Detection

In addition to the factors identified before, Full Disk encryption should be detectable through the sampling of multiple megabytes of data in various pseudo-random locations across the hard drive and a subsequent frequency analysis of clear-text strings in these samples. Unencrypted Operating Systems exhibit considerable numbers of plain text strings (e.g. in user files, linker libraries etc.) while an encrypted hard drive should exhibit drastically fewer strings. Samples were obtained from several encrypted and unencrypted hard drives.

3.2 Test Framework

In order to test the five hypotheses, a sample implementation for Microsoft Windows was developed. The resulting framework, termed ALERT (Automated Live Encryption Recognition Tool), integrates all of the tests listed below (with the exception of file hashes) with a simple-to-use Graphical User Interface. It runs the various tests in sequence and logs the start time, results and stop time for each test. It uses a signature file that contains all factors by which a particular program can be identified (see Table 2). Additionally, the Modified/Accessed/Created (MAC) times are saved before and after the execution to allow conclusions about the changes to MAC times resulting from the Live analysis. The framework was developed for users with minimal technical knowledge. It only requires a small number of options to be set by the user. The drive to be examined must be chosen from a drop-down menu. In addition, the user has to choose one of two modes of operation. The concise quick-scan performs all tests except for the file-header analysis. The full-scan option is more thorough but potentially very time-consuming.

Once the examination has been completed, the user is presented with four categories in which possible indicators of encryption software may fall as well as a tab that presents basic information about the computer and Operating System in use as well as some volatile information. This information is recorded to preserve as much information about the computer at the time of the analysis as possible as this information may become inaccessible if the computer is rebooted or used even if no encryption software is used. The user is presented with a color-coded dialog box that informs him whether the computer is exhibiting sure or probable signs of encryption software. All information is saved in a log file on a separate USB drive. This drive is also used to store the MAC times for all files on the examined drive. Together, these two files permit a forensic examiner to draw conclusions about the use of the computer without even having access to the contents of the drive.

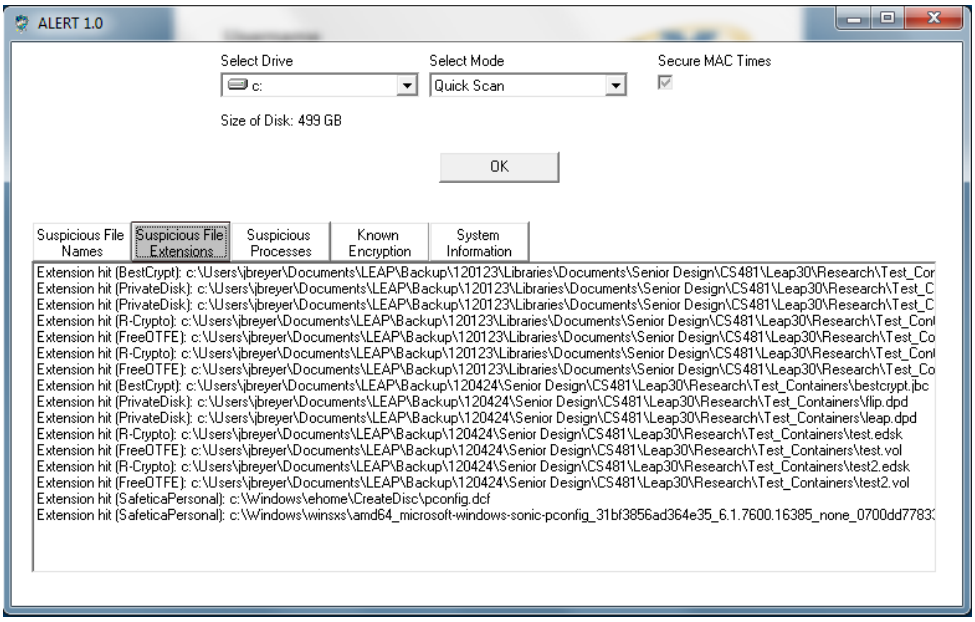


Figure 4 - ALERT Encryption Recognition Tool

3.2.1 Windows Registry Analysis

The Registry is a small, embedded database that has been part of every Microsoft Windows Operating System. It is used to store information about the current configuration of the Windows instance as well as the software installed in it. Virtually every program that is installed on a Windows computer stores key information relating to its configuration in the Registry. By examining the Registry for a key or value known to belong to a particular program one can very reliably deduce that the software was present at least at some point in the past. Programs are supposed to remove their keys during uninstallation. However, this is often not performed adequately and the presence of the key therefore does not necessarily indicate that the software is still present.

No information is available about how the registry database is implemented. The algorithm used for this framework can be assumed to require $\Theta(n)$ operations where n is the number of keys to be queried.

3.2.2 File Headers

Many file formats use a short sequence of constant values at the beginning of the file to enable consistency checks between the file's file extension and its content. This is, however, not a requirement and many non-standardized file formats do not utilize a constant header. In fact, many encryption programs deliberately avoid the use of a header to make it more difficult to identify a file as an encrypted container. When used consistently by a program, file headers are a good identifying factor but the fact that not many of the identified programs utilize them and that this is a very time-consuming check (every file on the drive must be opened for read-access and its first bytes compared to every signature in a file of known signatures) make it a non-optimal choice as a sole factor. The algorithm used for this framework can be assumed to require $\Theta(k*m)$ operations where k is the number of signature values and m is the number of files in the file system to be queried. This can take an extremely long time for a large file system.

3.2.3 File Extensions

In most modern operating systems, file names consist of a variable length name followed by a period character and a usually three to four character long file extension. In many cases, this file extension identifies the file as one of a particular format (e.g. docx for Microsoft's XML Word Document Format). Most of the identified encryption programs allow the user to choose any file extension for an encrypted container or none at all. Additionally, the file extension can usually be changed without consequences for the functionality of the program. This examination can be performed in a slightly faster fashion than the File Header analysis because most modern file systems treat file names as meta-information that is stored in a very efficient data structure that allows for fast access to this information without the necessity to traverse the relatively slow mechanical drive.

The algorithm used for this framework can be assumed to require Θ(k*m) operations where k is the number of signature values and m is the number of files in the file system to be queried.

3.2.4 File Attributes

File attributes, much like the File Name, are considered meta-information – information about the file. Common file attributes in current Operating Systems are the “hidden”,”write-protect” and “archive” flags. Microsoft Encrypting File System (EFS) utilizes this technique to flag files as “Encrypted”. EFS itself is simply a NTFS implementation that utilizes encryption for particular files or folders. In Microsoft Operating Systems, this test is very limited because no other program uses flag values to indicate encrypted files. However, this test is fairly fast (especially when combined with tests for other meta-information), reliable and it is entirely possible that other Operating or File Systems use this technique more extensively.

The algorithm used for this framework can be assumed to require Θ(m) operations where m is the number of files in the file system to be queried.

3.2.5 Master Boot Record

The Master Boot Record (MBR) is the boot code in the first 512 bytes of a hard drive.It contains some assembly code as well as signature values. Many of the Full Disk Encryption programs that were analyzed for this experiment stored an ASCII text-string or signature value in the MBR. In theory, this value can be overwritten without consequences but this requires at least moderate technical knowledge and, more importantly, knowledge that this signature value exists in the Master Boot Record. Since the amount of data that must be retrieved for this test and the

number of comparisons to be made are relatively small, this test is a fast and reliable indicator for the presence of Full Disk Encryption.

The algorithm used for this framework can be assumed to require $\Theta(k)$ operations where k is the number of signature values.

3.2.6 Operating System Attributes

With Windows Vista, Microsoft introduced an encryption mechanism called BitLocker that utilizes the presence of a hardware Trusted Platform Module (TPM) chip to encrypt the hard drive. Since the TPM chip is a part of the motherboard and cannot easily be moved, the hard drive becomes inaccessible if it is moved outside of the computer with which it was encrypted. Windows provides a convenient way of querying not only the use of BitLocker but also what its current state is (encrypting, decrypting, encrypted or decrypted). As it is implemented and maintained by the Operating System, this factor is very fast and reliable but can only be used to identify BitLocker.

The algorithm used for this framework can be assumed to require $\Theta(k*m)$ operations where k is the number of signature values and m is the number of files in the file system to be queried.

3.2.7 Keyword Search

Many encryption files have the words “crypt”, “safe” or “private” (or variations thereof) in their name or use these in file names. By searching for files that contain these substrings in their file name, programs that are unidentified or cannot be identified by other means may be found. Using the same mechanism as for the File Extension and File Attribute analyses, the process is relatively fast (however, due to the large number of comparisons slightly less so than

the other two meta-factors). It does produce large numbers of false positives as many off-the-shelf software products also contain files that fit the above pattern but it can help identify files and programs that may otherwise remain unnoticed. This is one of two checks employed by most of the other forensic software packages that claim to perform encryption detection.

The algorithm used for this framework can be assumed to require Θ(k*m) operations where k is the number of keywords and m is the number of files in the file system to be queried.

3.2.8 Process List Keyword Search

In Windows, Unix, Linux and MacOS, the process list contains the name and some other information about all non-kernel level programs that are currently running in the Operating System. Relying on the same keywords used in 3.3.7, many identified and unidentified encryption programs can be identified. Additionally, with the exception of BitLocker, there is usually no other way to identify whether or not the program is currently running. The benefits of this are assumed to outweigh the relatively modest number of false positives this method may yield.

The algorithm used for this framework can be assumed to require Θ(k*p) operations where k is the number of keywords and p is the number of processes currently running.

3.2.9 Cryptographic Hashing of Program Components

For this factor, a cryptographic checksum is computed for every file that the software is comprised of. If an appropriate hashing algorithm (such as SHA-256) is used, the likelihood of two files producing the same checksum is approaching zero.

While this method is extremely accurate, it suffers from two grave problems: every time a file is changed, the checksum changes which necessitates the computation of new reference

values. Secondly, the time to compute a cryptographic hash of every file in the File System as well as the time required for the comparison against the signature database could potentially take a very long time. For this reason, this factor was not included in ALERT.

3.2.10 Heuristic Detection

As described before, the basic properties of encryption suggest that one should be able to detect the use of encryption by examining collections of a few megabytes each from various places across the hard drive and analyzing the number of plain text strings in the samples. Theoretically, an encrypted hard drive should exhibit few, if any, plain text strings while they should be relatively abundant on a non-encrypted drive.

This method has the benefit that the software used to encrypt a full volume or hard drive does not even have to be present on the drive and the encryption could still be detected. Unfortunately, experiments conducted for this thesis showed that this is infeasible in practice as it clashes with another common property of current Full Disk Encryption: transparency. As long as the computer is in use, the presence of the encryption is completely transparent to both, the user and Operating System. The samples drawn from the test systems were virtually identical regardless of whether or not the volume was encrypted.

If one was able to examine the drive through a hypervisor that runs underneath the encryption software (e.g. in a Virtual Machine) or during a dead analysis, the contents of the hard drive are expected to confirm the hypothesis that encrypted drives can be identified by the low number of plain text strings observed on the drive. Since this work is geared towards Live Analysis, however, no further attempts in this direction were made and the analysis tools were not included in the ALERT framework.

3.3 Experiments

The detectability of encryption software was tested through three different sets of experiments. First, baseline experiments were conducted in which the test framework was run against a clean installation of Microsoft Windows XP, Windows 7 and Windows 8 as shown in table 3.

Next, the test framework was used to examine 18 installations of Microsoft Windows each of which contained one or more encrypted containers or hard disks. Programs for which a signature was available to the test framework were used to create the encrypted containers and partitions. Table 4 shows the exact configuration for each of the 18 experiments.

Last, tests were conducted approximately twelve months after the initial experiment. Each installation contained one of the previously identified programs but in the then-current version as listed in table 5.

To ensure the accuracy of the results, two trials were conducted for each experiment. The deviation between the experiments was recorded if the results between the two trials differed.

	Baseline Experiment 1	Baseline Experiment 2	Baseline Experiment 3
Hardware	Intel Core i5 3.30 GHz CPU, 8GB of RAM, 500GB Seagate HDD		
Operating System	Microsoft Windows XP SP 3	Microsoft Windows 7 SP 1	Microsoft Windows 8
Encryption Software	none	none	none

Table 3 - Baseline Experiments

	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6	Exp 7	Exp 8	Exp 9	Exp 10	Exp 11	Exp 12	Exp 13	Exp 14	Exp 15	Exp 16	Exp 17
Hardware	Intel Core i5 3.30 GHz CPU, 8GB of RAM, 500GB Seagate HDD																
Operating System	Microsoft Windows XP SP 3																Microsoft Windows 7 SP 1
Advanced File Security														✓		✓	
ArchiCrypt Live 5		✓	✓	✓	✓	✓											
BestCrypt			✓				✓										
Compusec															✓	✓	
CryptArchiveLite						✓											
Cypherix LE					✓	✓											
FreeOTFE										✓							
Keyparc												✓				✓	
Kruptos 2 Professional													✓				
LoopAES							✓	✓	✓	✓							
MyWinLocker				✓													
PGP Disk															✓	✓	
PrivateDisk											✓	✓				✓	
R-Crypto									✓	✓							
Safetica Personal											✓	✓				✓	
Sentry2020										✓							
TrueCrypt	✓																✓ (FDE)
A checkmark (✓) denotes that the software or an encrypted container created with it was present in the experiment.																	

Table 4 - Experiments against Software with Known Signature

	Exp 18	Exp 19	Exp 20	Exp 21	Exp 22	Exp 23	Exp 24	Exp 25
Hardware	Intel Core i7 2.9GHz CPU, 500GB HDD, 8GB RAM							
Operating System	Microsoft Windows 7 SP 1							
Advanced File Security	✓							
ArchiCrypt Live 5		✓						
BestCrypt			✓					
Microsoft BitLocker				✓				
PGP Disk								✓
PrivateDisk								
PrivateSafe HD					✓			
R-Crypto						✓		
TrueCrypt							✓ (FDE)	
A checkmark (✓) denotes that the software or an encrypted container created with it was present in the experiment.								

Table 5 - Experiments after 12 Months without Updated Signatures

Chapter 4 – Results

The following ten tables list the results for each of the 25 experiments. The first seven tables show the results obtained when using a particular factor in isolation. The final three tables show the results obtained with three different combinations of factors.

For each experiment, each table contains the number of True Positives, False Positives, True Negatives and False Negatives that were obtained as well as the Accuracy, Precision and Recall values that were calculated based on these results.

If there had been any differences between the results obtained in each of the two trials for each experiment, these would be noted in the table as well. Finally, the mean Precision, Recall and Accuracy values for the experiment and the standard deviation for each of the three values can be found at the end of each table.

Following are the results obtained for each of the experiments listed in Section 3.3 for the analysis of the Windows Registry.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	0	0						
Base 2	0	30	0	0						
Base 3	0	30	0	0						
Exp 1	1	29	0	0	1	0	1	0	1	0
Exp 2	1	29	0	0	1	0	1	0	1	0
Exp 3	2	28	0	0	1	0	1	0	1	0
Exp 4	2	28	0	0	1	0	1	0	1	0
Exp 5	2	28	0	0	1	0	1	0	1	0
Exp 6	3	27	0	0	1	0	1	0	1	0
Exp 7	2	28	0	0	1	0	1	0	1	0
Exp 8	1	29	0	0	1	0	1	0	1	0
Exp 9	2	28	0	0	1	0	1	0	1	0
Exp 10	4	26	0	0	1	0	1	0	1	0
Exp 11	2	28	0	0	1	0	1	0	1	0
Exp 12	3	27	0	0	1	0	1	0	1	0
Exp 13	1	29	0	0	1	0	1	0	1	0
Exp 14	1	29	0	0	1	0	1	0	1	0
Exp 15	2	28	0	0	1	0	1	0	1	0
Exp 16	6	24	0	0	1	0	1	0	1	0
Exp 17	1	29	0	0	1	0	1	0	1	0
Exp 18	1	29	0	0	1	0	1	0	1	0
Exp 19	1	29	0	0	1	0	1	0	1	0
Exp 20	0	29	0	1	0	0	0.96	0	0	0
Exp 21	0	29	0	1	0	0	0.96	0	0	0
Exp 22	1	29	0	0	1	0	1	0	1	0
Exp 23	1	29	0	0	1	0	1	0	1	0
Exp 24	1	29	0	0	1	0	1	0	1	0
Exp 25	0	29	0	1	0	0	0.96	0	0	0
Avg.					0.8800	0.3250	0.9952	0.0130	0.8800	0.3250

Table 6 - Results for Windows Registry Analysis

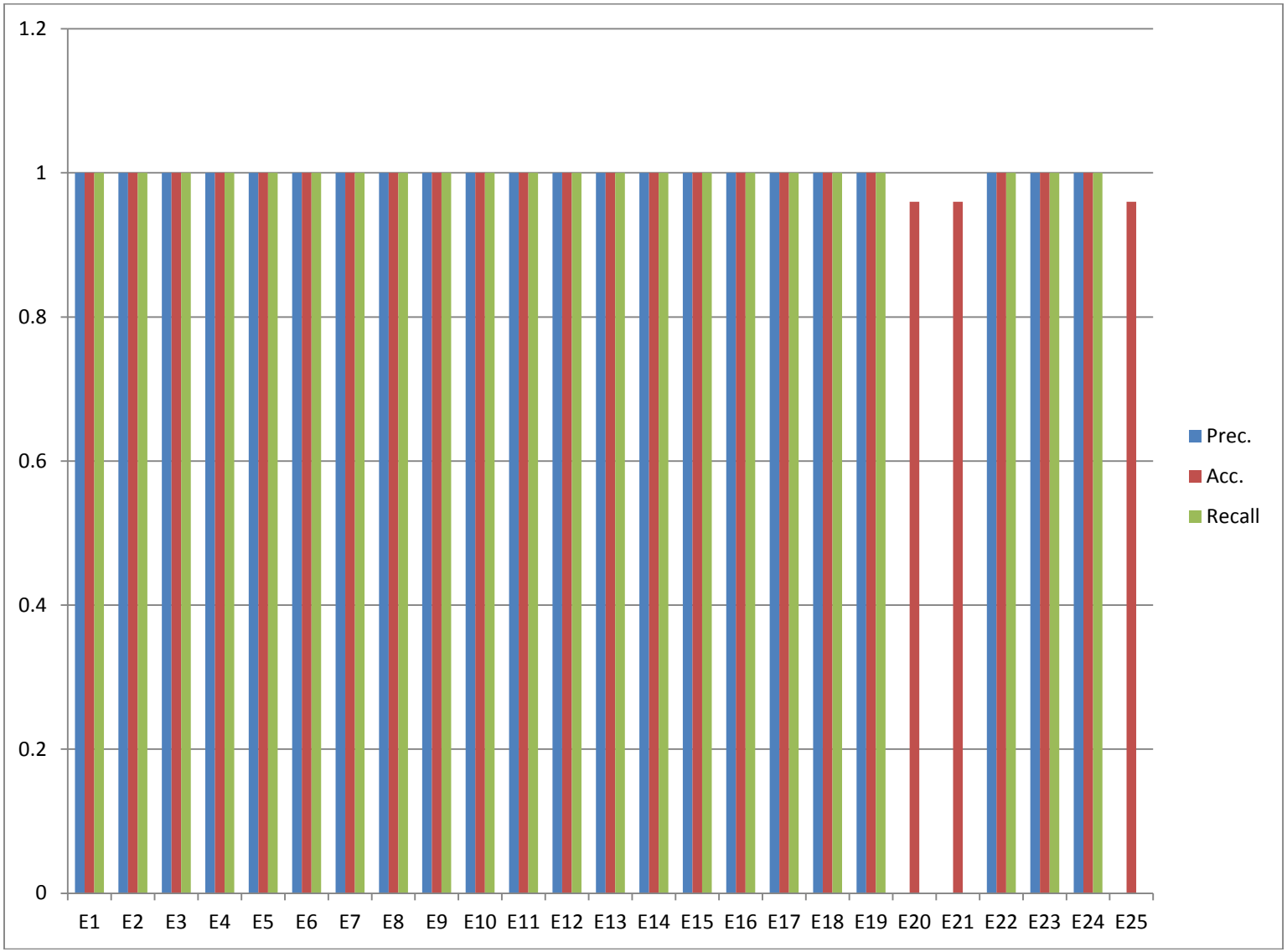


Figure 5 – Accuracy, Precision and Recall for the Windows Registry

Following are the results obtained for each of the experiments listed in Section 3.3 for the analysis of File Headers.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	0	0						
Base 2	0	30	0	0						
Base 3	0	30	0	0						
Exp 1	1	29	0	0	1	0	1	0	1	0
Exp 2	1	29	0	0	1	0	1	0	1	0
Exp 3	2	27	1	0	0.660	0	0.966	0	1	0
Exp 4	0	27	1	2	0	0	0.900	0	0	0
Exp 5	0	27	1	2	0	0	0.900	0	0	0
Exp 6	2	26	1	1	0.660	0	0.930	0	0.500	0
Exp 7	1	27	1	1	0.500	0	0.930	0	0.500	0
Exp 8	1	27	1	0	0.500	0	0.960	0	1	0
Exp 9	1	27	1	1	0.500	0	0.930	0	0.500	0
Exp 10	2	25	1	2	0.660	0	0.900	0	0.500	0
Exp 11	0	27	1	2	0	0	0.900	0	0	0
Exp 12	0	26	1	3	0	0	0.860	0	0	0
Exp 13	0	28	1	1	0	0	0.900	0	0	0
Exp 14	1	29	0	0	1	0	1	0	1	0
Exp 15	1	27	1	1	0.500	0	0.900	0	0.500	0
Exp 16	1	25	1	5	0.500	0	0.800	0	0.160	0
Exp 17	0	29	0	1	0	0	0.966	0	0	0
Exp 18	0	29	0	1	0	0	0.966	0	0	0
Exp 19	0	29	0	1	0	0	0.966	0	0	0
Exp 20	1	29	0	0	1	0	1	0	1	0
Exp 21	0	29	0	1	0	0	0.966	0	0	0
Exp 22	0	29	0	1	0	0	0.966	0	0	0
Exp 23	0	29	0	1	0	0	0.966	0	0	0
Exp 24	0	29	0	1	0	0	0.966	0	0	0
Exp 25	0	29	0	1	0	0	0.966	0	0	0
Avg.					0.2992	0.3837	0.9374	0.0477	0.2992	0.4136

Table 7 - Results for File Header Analysis

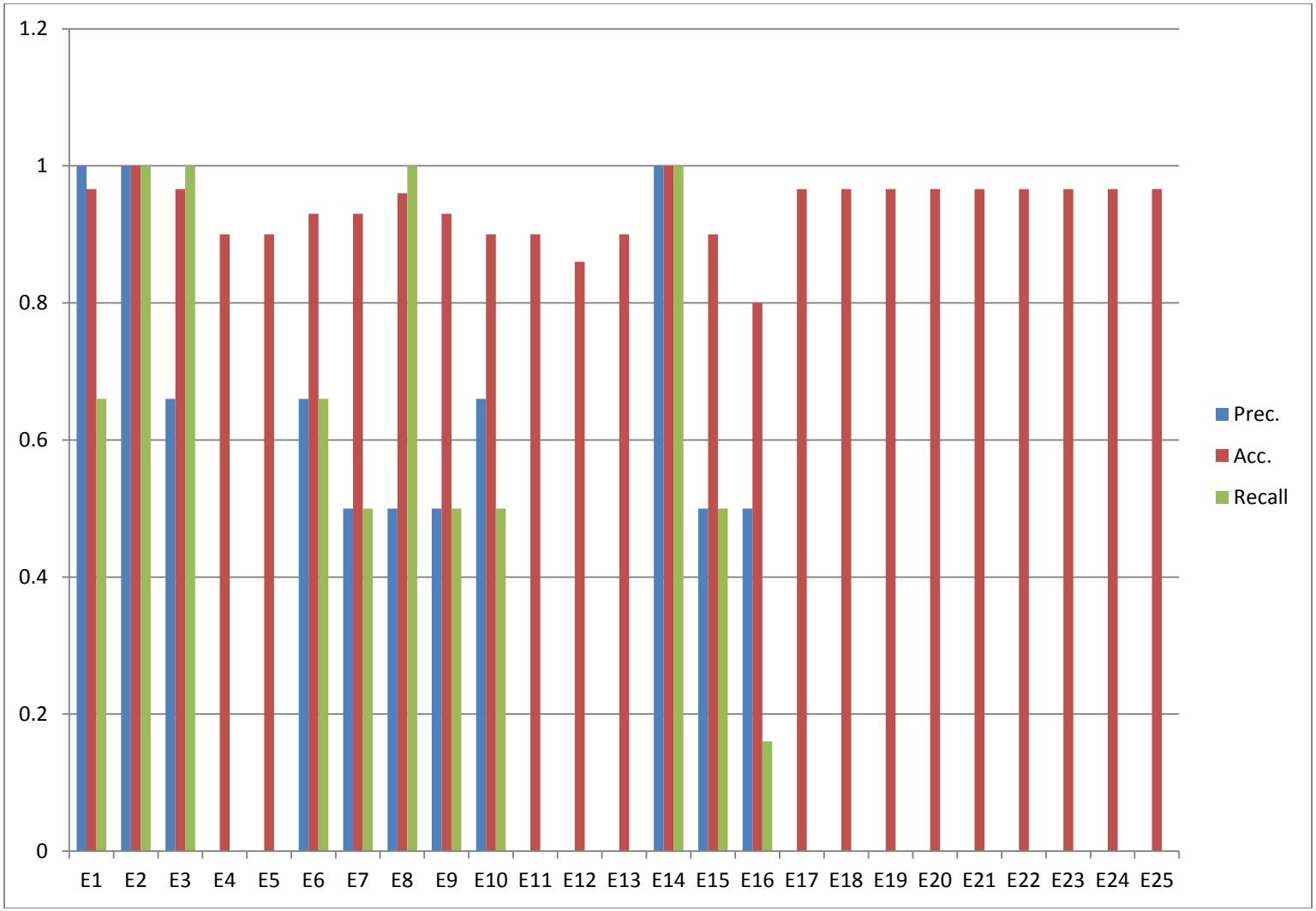


Figure 6 - Precision, Accuracy and Recall for File Headers

Following are the results obtained for each of the experiments listed in Section 3.3 for the analysis of File Extensions.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	0	0						
Base 2	0	30	0	0						
Base 3	0	30	0	0						
Exp 1	0	29	0	1	0	0	0.966	0	0	0
Exp 2	0	29	0	1	0	0	0.966	0	0	0
Exp 3	1	28	0	1	1	0	0.930	0	0.500	0
Exp 4	0	28	0	2	0	0	0.930	0	0	0
Exp 5	1	28	0	1	1	0	0.930	0	0.500	0
Exp 6	0	27	0	3	0	0	0.900	0	0	0
Exp 7	2	28	0	0	1	0	1	0	1	0
Exp 8	1	29	0	0	1	0	1	0	1	0
Exp 9	2	28	0	0	1	0	1	0	1	0
Exp 10	3	26	0	1	1	0	0.930	0	0.750	0
Exp 11	1	28	0	1	1	0	0.930	0	0.500	0
Exp 12	0	27	0	3	0	0	0.900	0	0	0
Exp 13	1	29	0	0	1	0	1	0	1	0
Exp 14	1	29	0	0	1	0	1	0	1	0
Exp 15	1	28	0	1	1	0	0.930	0	0.500	0
Exp 16	1	24	0	5	1	0	0.800	0	0.160	0
Exp 17	0	29	0	1	0	0	0.966	0	0	0
Exp 18	1	29	2	0	0.33	0	0.9375	0	1	0
Exp 19	0	29	2	1	0	0	0.9063	0	0	0
Exp 20	1	29	2	0	0.33	0	0.9375	0	1	0
Exp 21	0	29	2	1	0	0	0.9063	0	0	0
Exp 22	0	29	2	1	0	0	0.9063	0	0	0
Exp 23	0	29	2	1	0	0	0.9063	0	0	0
Exp 24	1	29	2	0	0.33	0	0.9375	0	1	0
Exp 25	0	29	2	1	0	0	0.9063	0	0	0
Avg.					0.4796	0.4723	0.9366	0.0442	0.4364	0.4394

Table 8 - Results for File Extension Analysis

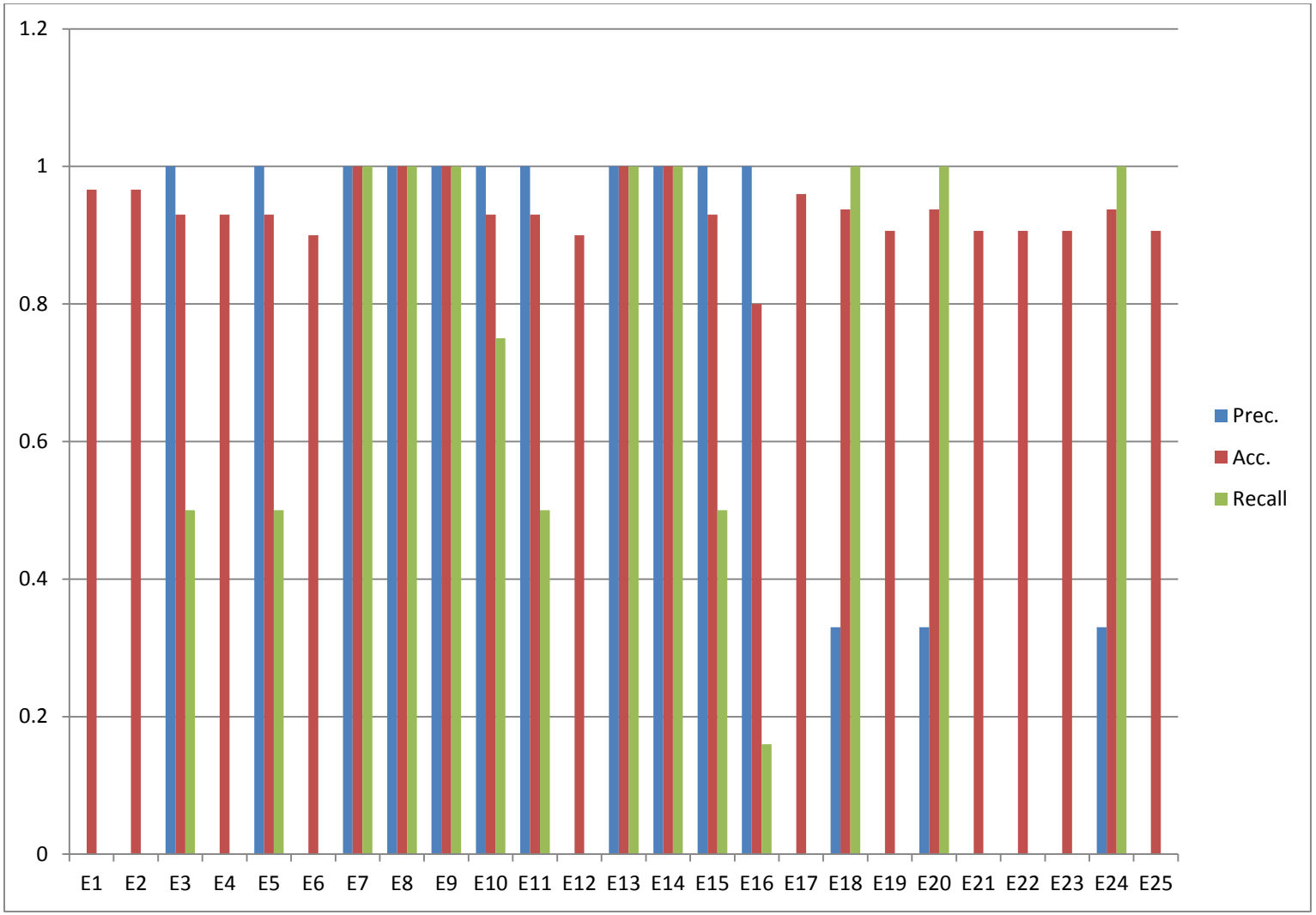


Figure 7 - Precision, Accuracy and Recall for File Extensions

Following are the results obtained for each of the experiments listed in Section 3.3 for the analysis of File Attributes.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	0	0						
Base 2	0	30	0	0						
Base 3	0	30	0	0						
Exp 1	0	29	0	1	0	0	0.966	0	0	0
Exp 2	0	29	0	1	0	0	0.966	0	0	0
Exp 3	0	28	0	2	0	0	0.930	0	0	0
Exp 4	0	28	0	2	0	0	0.930	0	0	0
Exp 5	0	28	0	2	0	0	0.930	0	0	0
Exp 6	0	27	0	3	0	0	0.900	0	0	0
Exp 7	0	28	0	2	0	0	0.930	0	0	0
Exp 8	0	29	0	1	0	0	0.930	0	0	0
Exp 9	0	28	0	2	0	0	0.930	0	0	0
Exp 10	0	26	0	4	0	0	0.860	0	0	0
Exp 11	0	28	0	2	0	0	0.930	0	0	0
Exp 12	0	27	0	3	0	0	0.900	0	0	0
Exp 13	0	29	0	1	0	0	0.960	0	0	0
Exp 14	0	29	0	1	0	0	0.960	0	0	0
Exp 15	0	28	0	2	0	0	0.930	0	0	0
Exp 16	0	24	0	6	0	0	0.800	0	0	0
Exp 17	0	29	0	1	0	0	0.966	0	0	0
Exp 18	0	29	0	1	0	0	0.966	0	0	0
Exp 19	0	29	0	1	0	0	0.966	0	0	0
Exp 20	0	29	0	1	0	0	0.966	0	0	0
Exp 21	0	29	0	1	0	0	0.966	0	0	0
Exp 22	0	29	0	1	0	0	0.966	0	0	0
Exp 23	0	29	0	1	0	0	0.966	0	0	0
Exp 24	0	29	0	1	0	0	0.966	0	0	0
Exp 25	0	29	0	1	0	0	0.966	0	0	0
Avg.					0	0	0.9356	0.0390	0	0

Table 9 - Results for File Attribute Analysis

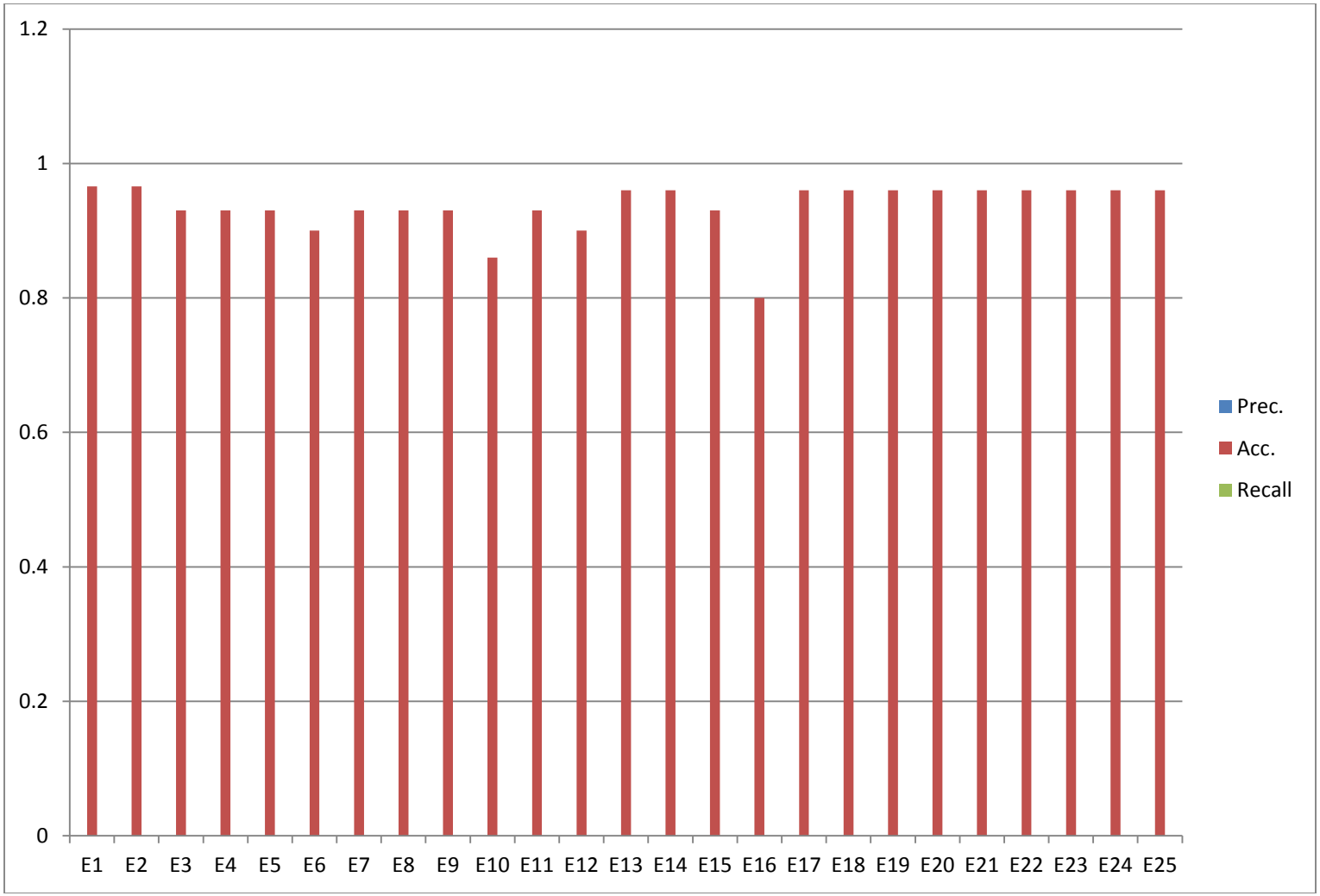


Figure 8 - Precision, Accuracy and Recall for File Attributes

*none of the test cases utilized EFS leading to a constant zero-detection rate

Following are the results obtained for each of the experiments listed in Section 3.3 for the analysis of the Master Boot Record.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	0	0						
Base 2	0	30	0	0						
Base 3	0	30	0	0						
Exp 1	1	29	0	0	1	0	1	0	1	0
Exp 2	0	29	0	1	0	0	0.966	0	0	0
Exp 3	1	28	0	1	1	0	0.930	0	0.500	0
Exp 4	0	28	0	2	0	0	0.930	0	0	0
Exp 5	0	28	0	2	0	0	0.930	0	0	0
Exp 6	0	27	0	3	0	0	0.900	0	0	0
Exp 7	1	28	0	1	1	0	0.966	0	0.500	0
Exp 8	0	29	0	1	0	0	0.966	0	0	0
Exp 9	0	28	0	2	0	0	0.930	0	0	0
Exp 10	0	26	0	4	0	0	0.860	0	0	0
Exp 11	0	28	0	2	0	0	0.930	0	0	0
Exp 12	0	27	0	3	0	0	0.900	0	0	0
Exp 13	0	29	0	1	0	0	0.966	0	0	0
Exp 14	0	29	0	1	0	0	0.966	0	0	0
Exp 15	0	28	0	2	0	0	0.930	0	0	0
Exp 16	1	24	0	5	1	0	0.800	0	0.160	0
Exp 17	1	29	0	0	1	0	1	0	1	0
Exp 18	0	29	0	1	0	0	0.966	0	0	0
Exp 19	0	29	0	1	0	0	0.966	0	0	0
Exp 20	1	29	0	0	1	0	1	0	1	0
Exp 21	0	29	0	1	0	0	0.966	0	0	0
Exp 22	0	29	0	1	0	0	0.966	0	0	0
Exp 23	0	29	0	1	0	0	0.966	0	0	0
Exp 24	0	29	0	1	0	0	0.966	0	0	0
Exp 25	0	29	0	1	0	0	0.966	0	0	0
Avg.					0.2400	0.4271	0.9443	0.0438	0.1680	0.3367

Table 10 - Results for Master Boot Record Analysis

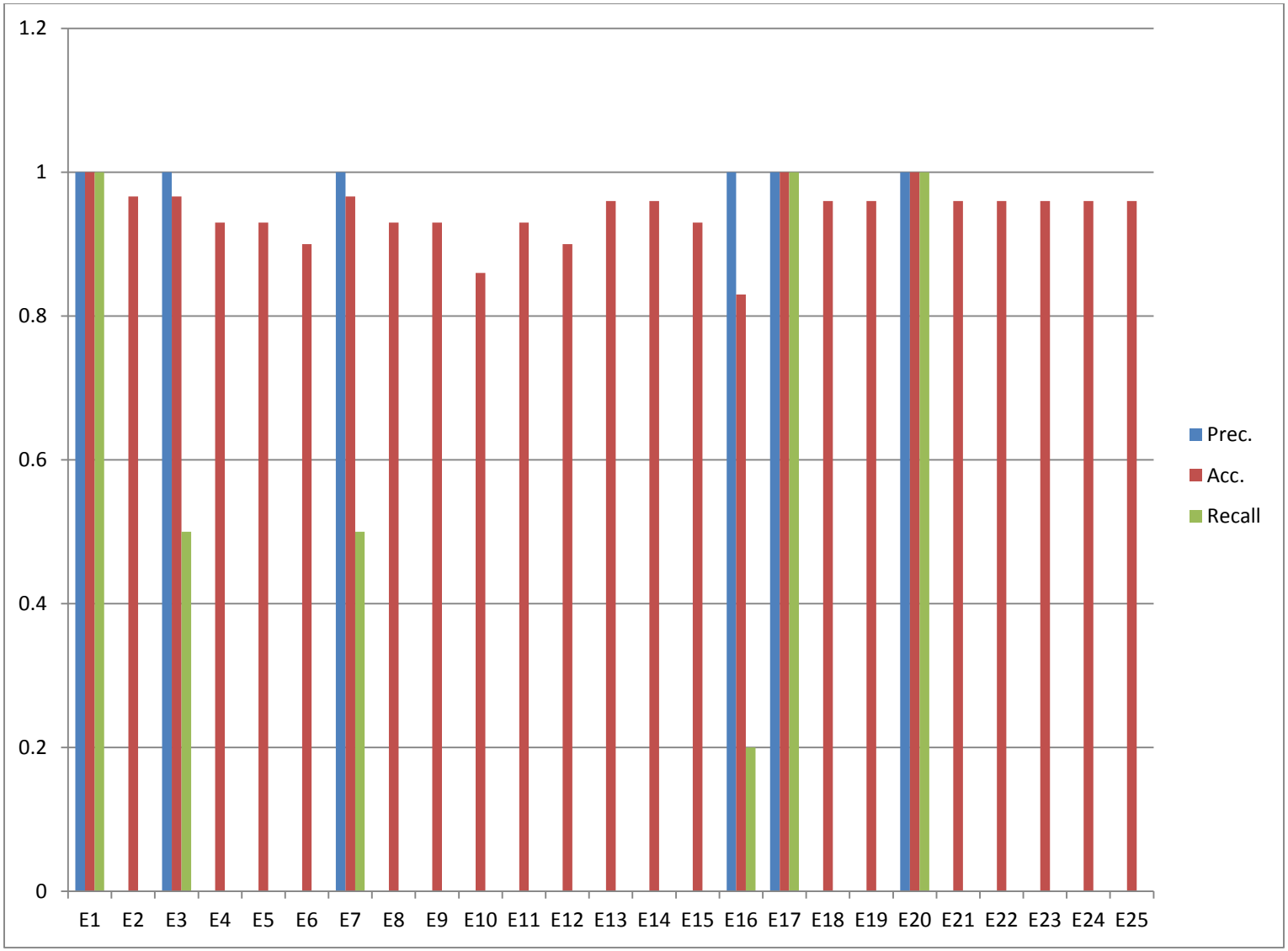


Figure 9 - Precision, Accuracy and Recall for the Master Boot Record

Following are the results obtained for each of the experiments listed in Section 3.3 for the analysis of Operating System Attributes.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	0	0						
Base 2	0	30	0	0						
Base 3	0	30	0	0						
Exp 1	0	29	0	1	0	0	0.966	0	0	0
Exp 2	0	29	0	1	0	0	0.966	0	0	0
Exp 3	0	28	0	2	0	0	0.930	0	0	0
Exp 4	0	28	0	2	0	0	0.930	0	0	0
Exp 5	0	28	0	2	0	0	0.930	0	0	0
Exp 6	0	27	0	3	0	0	0.900	0	0	0
Exp 7	0	28	0	2	0	0	0.930	0	0	0
Exp 8	0	29	0	1	0	0	0.966	0	0	0
Exp 9	0	28	0	2	0	0	0.930	0	0	0
Exp 10	0	26	0	4	0	0	0.860	0	0	0
Exp 11	0	28	0	2	0	0	0.930	0	0	0
Exp 12	0	27	0	3	0	0	0.900	0	0	0
Exp 13	0	29	0	1	0	0	0.960	0	0	0
Exp 14	0	29	0	1	0	0	0.960	0	0	0
Exp 15	0	28	0	2	0	0	0.930	0	0	0
Exp 16	0	24	0	6	0	0	0.800	0	0	0
Exp 17	0	29	0	1	0	0	0.966	0	0	0
Exp 18	0	29	0	1	0	0	0.966	0	0	0
Exp 19	0	29	0	1	0	0	0.966	0	0	0
Exp 20	0	29	0	1	0	0	0.966	0	0	0
Exp 21	1	29	0	0	1	0	1	0	1	0
Exp 22	0	29	0	1	0	0	0.966	0	0	0
Exp 23	0	29	0	1	0	0	0.966	0	0	0
Exp 24	0	29	0	1	0	0	0.966	0	0	0
Exp 25	0	29	0	1	0	0	0.966	0	0	0
Avg.					0.0400	0.1960	0.9373	0.0409	0.0400	0.1960

Table 11 - Results for Operating System Attribute Analysis

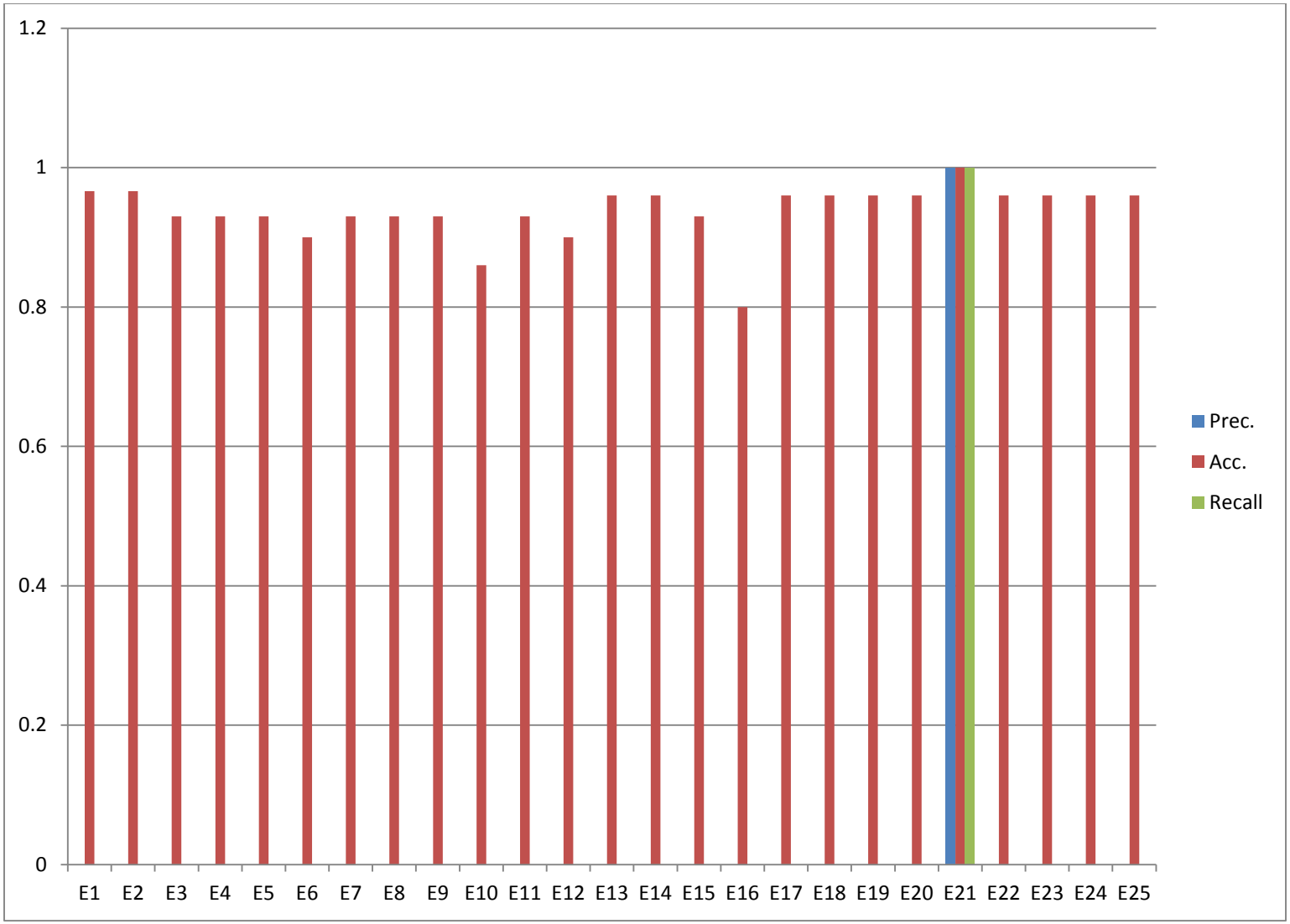


Figure 10 - Precision, Accuracy and Recall for OS Attributes

Following are the results obtained for each of the experiments listed in Section 3.3 for the analysis of Keywords.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	28	0						
Base 2	0	30	29	0						
Base 3	0	30	29	0						
Exp 1	1	29	38	0	0.026	0	0.441	0	1	0
Exp 2	1	29	21	0	0.047	0	0.588	0	1	0
Exp 3	2	28	22	0	0.091	0	0.566	0	1	0
Exp 4	2	28	22	0	0.091	0	0.566	0	1	0
Exp 5	1	28	26	1	0.037	0	0.509	0	0.500	0
Exp 6	2	27	29	1	0.065	0	0.491	0	0.666	0
Exp 7	1	28	22	1	0.046	0	0.558	0	0.500	0
Exp 8	0	29	32	1	0	0	0.468	0	0	0
Exp 9	1	28	29	1	0.033	0	0.491	0	0.500	0
Exp 10	2	26	23	2	0.087	0	0.528	0	0.500	0
Exp 11	0	28	23	2	0	0	0.528	0	0	0
Exp 12	0	27	31	3	0	0	0.443	0	0	0
Exp 13	0	29	32	1	0	0	0.468	0	0	0
Exp 14	0	29	32	1	0	0	0.468	0	0	0
Exp 15	0	28	31	2	0	0	0.459	0	0	0
Exp 16	2	24	40	4	0.050	0	0.371	0	0.333	0
Exp 17	1	29	28	0	0.034	0	0.517	0	1	0
Exp 18	0	29	292	1	0	0	0.089	0	0	0
Exp 19	1	29	292	0	0.003	0	0.093	0	1	0
Exp 20	1	29	292	0	0.003	0	0.093	0	1	0
Exp 21	0	29	292	1	0	0	0.089	0	0	0
Exp 22	1	29	292	0	0.003	0	0.093	0	1	0
Exp 23	0	29	292	1	0	0	0.089	0	0	0
Exp 24	1	29	292	0	0.003	0	0.093	0	1	0
Exp 25	0	29	292	1	0	0	0.089	0	0	0
Avg.					0.0247	0.0311	0.3684	0.1949	0.4799	0.4380

Table 12 - Results for Keyword Analysis

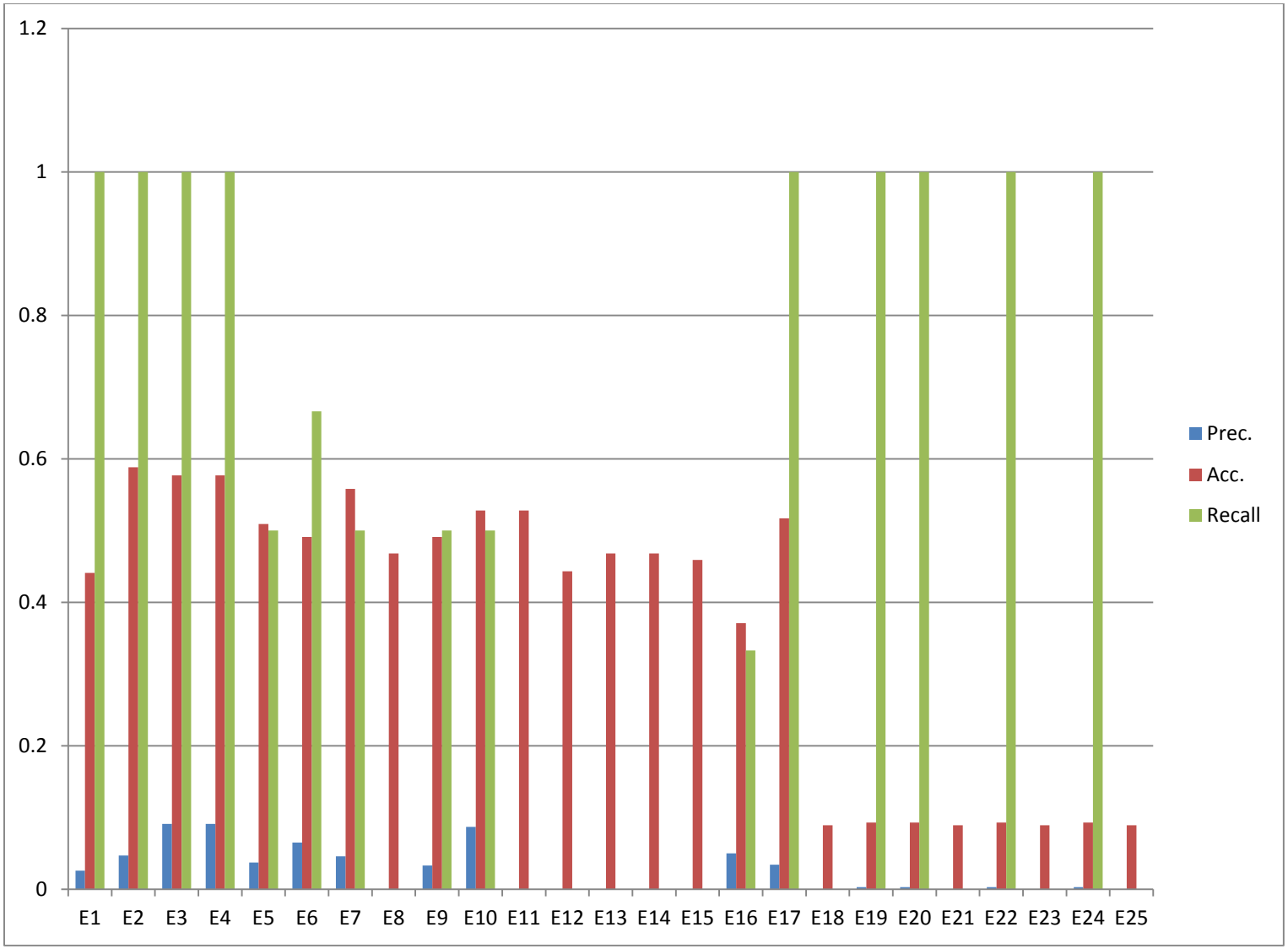


Figure 11 - Precision, Accuracy and Recall for Keyword Analysis

Following are the results obtained for each of the experiments listed in Section 3.3 for the first combination of different factors.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	28	0						
Base 2	0	30	29	0						
Base 3	0	30	29	0						
Exp 1	1	29	38	0	0.026	0	0.441	0	1	0
Exp 2	1	29	21	0	0.047	0	0.588	0	1	0
Exp 3	2	28	22	0	0.091	0	0.566	0	1	0
Exp 4	2	28	22	0	0.091	0	0.566	0	1	0
Exp 5	2	28	27	0	0.069	0	0.526	0	1	0
Exp 6	2	27	30	1	0.063	0	0.483	0	0.666	0
Exp 7	2	28	23	0	0.080	0	0.566	0	1	0
Exp 8	1	29	33	0	0.029	0	0.476	0	1	0
Exp 9	2	28	30	0	0.063	0	0.500	0	1	0
Exp 10	3	26	24	1	0.111	0	0.537	0	0.750	0
Exp 11	2	28	24	0	0.077	0	0.555	0	1	0
Exp 12	3	27	32	0	0.086	0	0.484	0	1	0
Exp 13	1	29	33	0	0.029	0	0.476	0	1	0
Exp 14	1	29	33	0	0.029	0	0.476	0	1	0
Exp 15	2	28	32	0	0.059	0	0.484	0	1	0
Exp 16	4	24	42	2	0.087	0	0.388	0	0.666	0
Exp 17	1	29	28	0	0.034	0	0.517	0	1	0
Exp 18	1	29	292	0	0.003	0	0.093	0	1	0
Exp 19	1	29	292	0	0.003	0	0.093	0	1	0
Exp 20	1	29	292	0	0.003	0	0.093	0	1	0
Exp 21	1	29	292	0	0.003	0	0.093	0	1	0
Exp 22	1	29	292	0	0.003	0	0.093	0	1	0
Exp 23	1	29	292	0	0.003	0	0.093	0	1	0
Exp 24	1	29	292	0	0.003	0	0.093	0	1	0
Exp 25	0	29	292	1	0	0	0.090	0	0	0
Avg.					0.0428	0.0354	0.3748	0.1980	0.9233	0.2134

Table 13 - Results for Combination 1 (Registry, File Extensions, File Attributes, MBR, OS Attributes and Keywords)

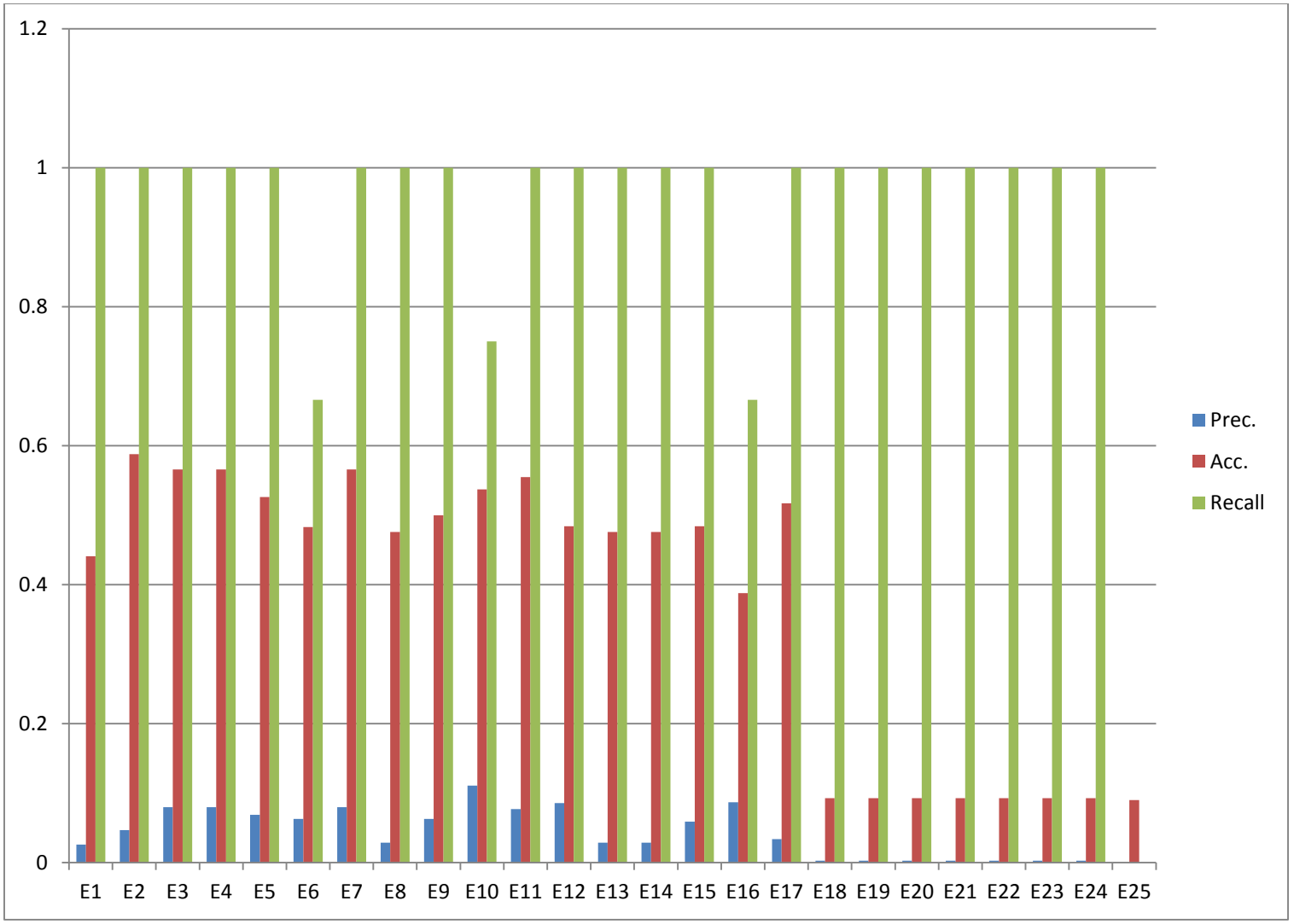


Figure 12 - Precision, Accuracy and Recall for Combination 1

Following are the results obtained for each of the experiments listed in Section 3.3 for the second combination of different factors.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	28	0						
Base 2	0	30	29	0						
Base 3	0	30	29	0						
Exp 1	1	29	38	0	0.026	0	0.441	0	1	0
Exp 2	1	29	21	0	0.047	0	0.588	0	1	0
Exp 3	2	28	23	0	0.080	0	0.566	0	1	0
Exp 4	2	28	23	0	0.080	0	0.566	0	1	0
Exp 5	2	28	27	0	0.069	0	0.526	0	1	0
Exp 6	3	27	30	0	0.091	0	0.500	0	1	0
Exp 7	2	28	23	0	0.080	0	0.566	0	1	0
Exp 8	1	29	33	0	0.029	0	0.476	0	1	0
Exp 9	2	28	30	0	0.063	0	0.500	0	1	0
Exp 10	4	26	24	0	0.143	0	0.555	0	1	0
Exp 11	2	28	24	0	0.077	0	0.555	0	1	0
Exp 12	3	27	32	0	0.086	0	0.484	0	1	0
Exp 13	1	29	33	0	0.029	0	0.476	0	1	0
Exp 14	1	29	33	0	0.029	0	0.476	0	1	0
Exp 15	2	28	32	0	0.059	0	0.484	0	1	0
Exp 16	5	24	42	1	0.106	0	0.403	0	0.833	0
Exp 17	1	29	28	0	0.034	0	0.517	0	1	0
Exp 18	1	29	292	0	0.003	0	0.093	0	1	0
Exp 19	1	29	292	0	0.003	0	0.093	0	1	0
Exp 20	1	29	292	0	0.003	0	0.093	0	1	0
Exp 21	1	29	292	0	0.003	0	0.093	0	1	0
Exp 22	1	29	292	0	0.003	0	0.093	0	1	0
Exp 23	1	29	292	0	0.003	0	0.093	0	1	0
Exp 24	1	29	292	0	0.003	0	0.093	0	1	0
Exp 25	0	29	292	1	0	0	0.090	0	0	0
Avg.					0.0459	0.0393	0.3768	0.1991	0.9533	0.1973

Table 14 - Results for Combination 2 (Registry, File Extensions, File Headers, File Attributes, MBR, OS Attributes, Keywords)

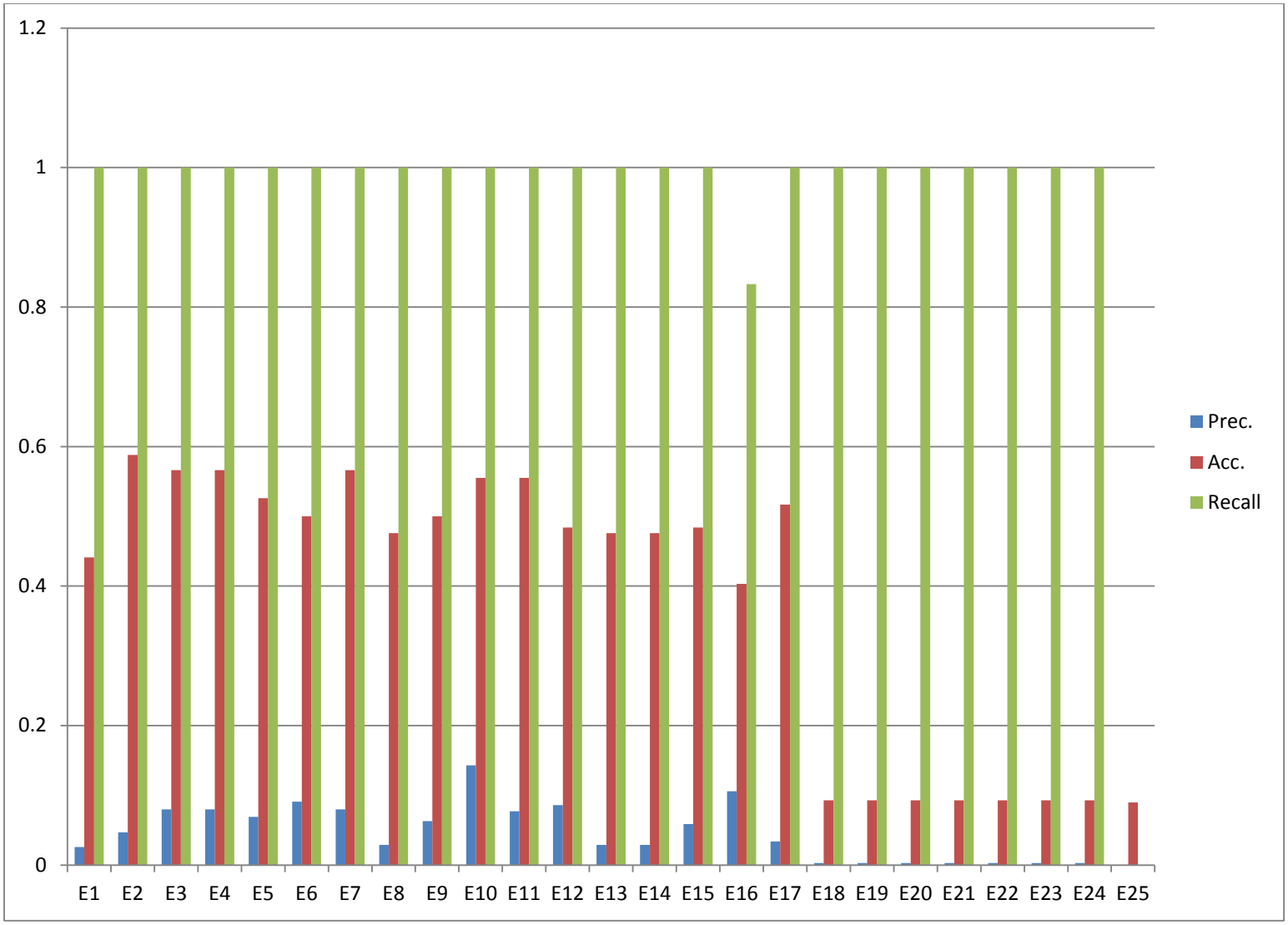


Figure 13 - Precision, Accuracy and Recall for Combination 2

Following are the results obtained for each of the experiments listed in Section 3.3 for the third combination of different factors.

	True Positives	True Negatives	Type I Error (False Positive)	Type II Error (False Negative)	Precision	Std. Dev.	Accuracy	Std. Dev.	Recall	Std. Dev.
Base 1	0	30	0	0						
Base 2	0	30	0	0						
Base 3	0	30	0	0						
Exp 1	1	29	1	0	0.500	0	0.968	0	1	0
Exp 2	1	29	0	0	1	0	1	0	1	0
Exp 3	2	28	0	0	1	0	1	0	1	0
Exp 4	2	28	0	0	1	0	1	0	1	0
Exp 5	2	28	0	0	1	0	1	0	1	0
Exp 6	2	27	0	1	1	0	0.966	0	0.666	0
Exp 7	2	28	0	0	1	0	1	0	1	0
Exp 8	1	29	0	0	1	0	1	0	1	0
Exp 9	2	28	0	0	1	0	1	0	1	0
Exp 10	3	26	0	0	1	0	0.966	0	0.750	0
Exp 11	2	28	0	0	1	0	1	0	1	0
Exp 12	3	27	0	0	1	0	1	0	1	0
Exp 13	1	29	0	0	1	0	1	0	1	0
Exp 14	1	29	0	0	1	0	1	0	1	0
Exp 15	2	28	0	0	1	0	1	0	1	0
Exp 16	5	24	0	1	1	0	0.966	0	0.833	0
Exp 17	1	29	0	0	1	0	1	0	1	0
Exp 18	1	29	0	0	1	0	1	0	1	0
Exp 19	1	29	0	0	1	0	1	0	1	0
Exp 20	1	29	0	0	1	0	1	0	1	0
Exp 21	1	29	0	0	1	0	1	0	1	0
Exp 22	1	29	0	0	1	0	1	0	1	0
Exp 23	1	29	0	0	1	0	1	0	1	0
Exp 24	1	29	0	0	1	0	1	0	1	0
Exp 25	0	29	0	1	0	0	0	0	0	0
Avg.					0.9400	0.2154	0.9546	0.1952	0.9299	0.2078

Table 15 - Results for Combination 3 (Registry, File Extensions, File Attributes, MBR and OS Attributes)

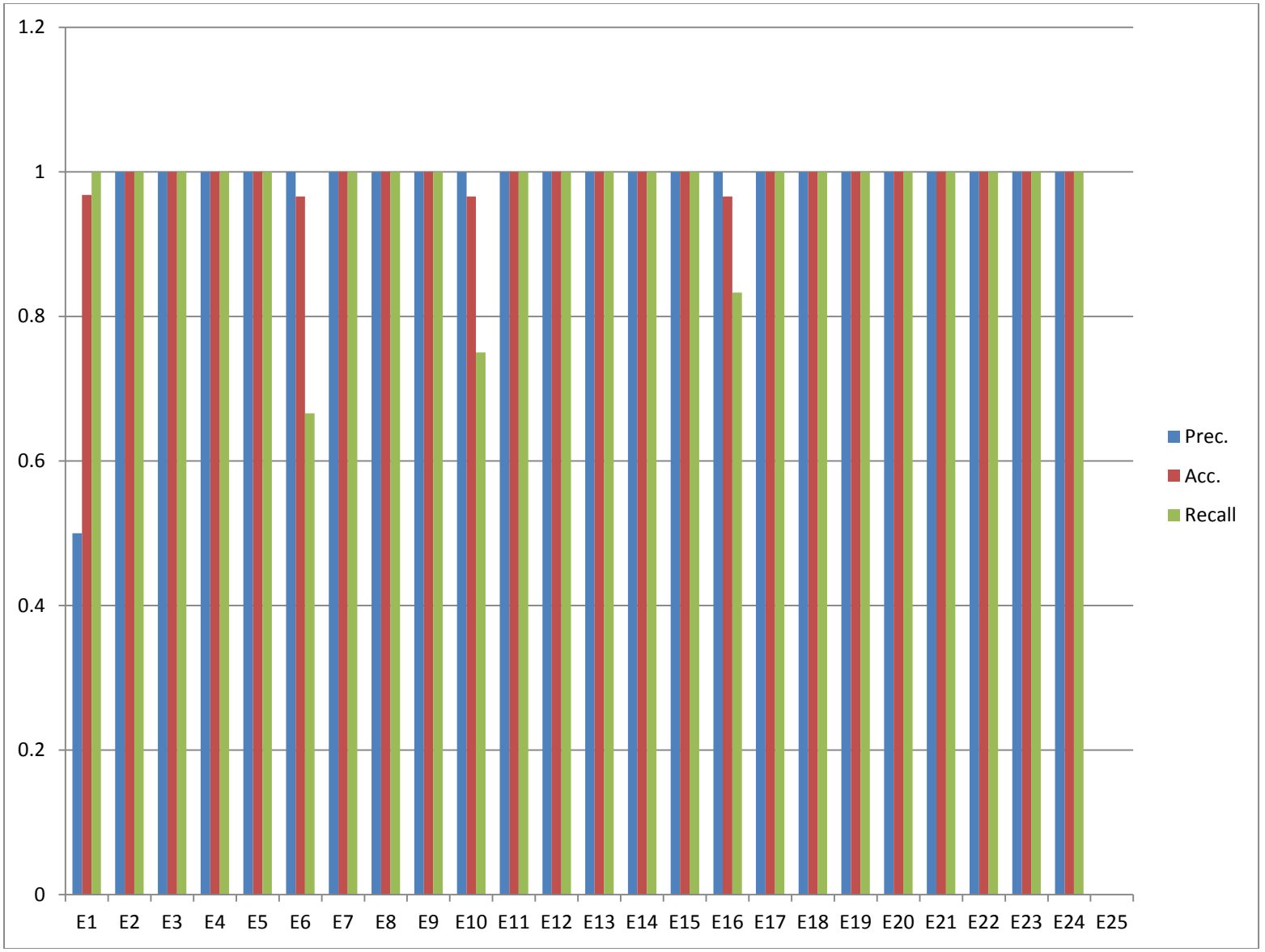


Figure 14 - Precision, Accuracy and Recall for Combination 3

Chapter 5 - Conclusion

The results obtained during the experiments suggest that it is indeed possible to detect the most common Windows encryption programs with a signature based approach. As can be seen in Table 4.9, a mean recall value of 0.95 was achieved with one of the combinations. This means that on average, 95% of the encryption programs, file systems or containers present in the test cases were detected.

Contrary to prediction 1.1.1, the inclusion of File Header analysis improves the results minutely, if at all, and increases the runtime exponentially. The null-hypothesis (H_0 = There exists one or more factor such that $recall_F >= recall_{header}$) must therefore be accepted.

Without the header analysis, all of the file system attributes can be analyzed without an additional increase in runtime rendering the exclusion of a particular attribute neutral in terms of execution time and negative in terms of recall.

Due to the lack of a heuristic detection method that can be executed during a live analysis (i.e. on a running computer), prediction 1.1.2 could not be tested in its entirety and consequently, no conclusions can be drawn about the performance of heuristic predictors. Contrary to prediction 1.1.2, the results suggest that the Keyword Analysis does not improve the results even if no signature for the particular version of the software is available. The null-hypothesis (H_0 = There exists a factor F such that $recall_F >= recall_{keyword}$ for software products for which no accurate signature exists) must therefore be accepted.

Additionally, the method has very high false positive rates that degrade precision and accuracy measures. While not necessarily negative in this context, false positives present a nuisance to the investigator that outweighs the little benefit this method provides.

As can be seen in Figures 15, 16 and 17, Registry Analysis is more reliable in terms of recall than any other single predictor and itself performs nearly as well as any combination of factors. Yet, the alternate hypothesis H_1 stated in section 1.1.3 narrowly holds up as the results obtained with combinations C_1-C_3 improve the detection rate for a small set of the trials. Based on the experiments, the most useful combination C includes the following factors: Windows Registry, Operating System Attributes, File Attributes, File Extensions, and Master Boot Record.

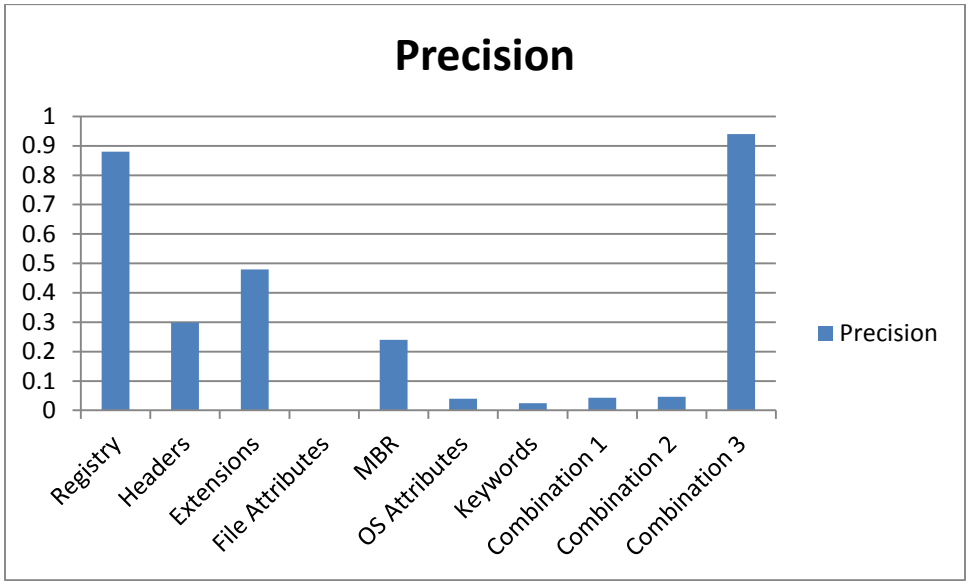


Figure 15 - Comparison of the Average Precision

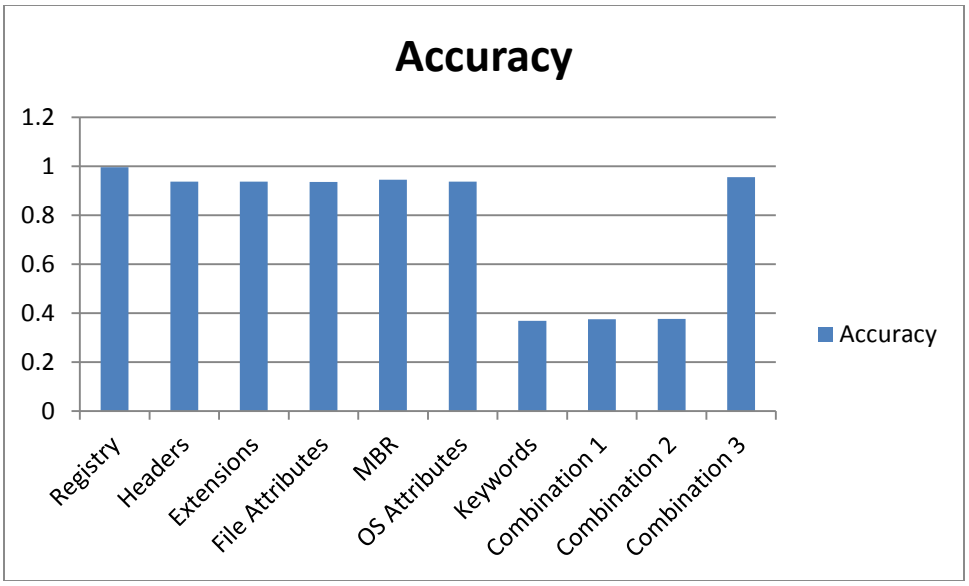


Figure 16 - Comparison of the Average Accuracy

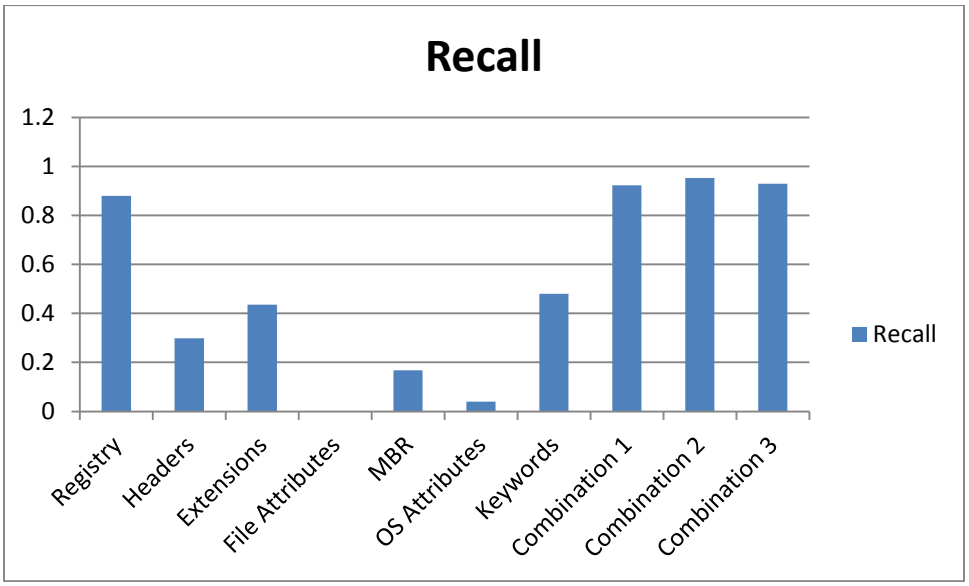


Figure 17 - Comparison of the Average Recall

Beyond the predictions made in Section 1.1, one can conclude that recall, precision and accuracy are all inadequate predictors for the utility of a particular detection factor. The Operating System Attribute factor, for example, only detects one method of encryption and therefore yields very low precision and recall values. However, it is the only means by which the

widely used encryption software Microsoft BitLocker can be detected and its practical utility is therefore very high while its execution time is negligible.

Due to the fact that accuracy is the ratio of the sum of true positives and true negatives to the sum of all four cells of the confusion matrix, its value is an even worse predictor. A factor yielding neither true nor false positives can, and often does, attain a high accuracy value but may be completely useless for the analysis.

Likewise, precision can be regarded as of very limited use because it considers only the ratio of true positives to the sum of true and false positives while ignoring false negatives entirely. As stated in the opening chapter of this examination, even elevated numbers of false positives may be acceptable for the task at hand if they are accompanied by a minimum of false negatives as these are typically catastrophic for the investigation.

Chapter 6 - Future Work

The framework that was developed as part of this work was targeted at the first responders of our law enforcement agencies. These responders often have limited understanding of technology and little access to specialists or specialized hardware and must determine whether a computer can or should be analyzed in a laboratory without the risk of loss of data or evidence. The tools provided to them should therefore minimize the amount of human intervention and the necessity for human analysis. An important area of future work is therefore the development or application of machine learning algorithms that can make reliable risk determinations based on the information gathered during the automated analysis. The ALERT framework in its current form can only draw on the type and number of indicia found to make a recommendation to the user. The first responder still has to review all of the results and decide if she arrives at the same determination as the analysis tool.

A second important area of focus is the development of heuristic predictors for the presence of encryption. These predictors should be statistically verifiable. As shown in section 5, most of the factors identified for this analysis work dependably only as long as a signature for the exact version of the software exists. Anti-Virus software suffers from the same necessity to constantly maintain a signature database and companies have developed alternate means that observe the behavior of a process or program to identify possibly malign behavior. Similar means are needed for the identification and detection of encryption programs as well.

Lastly, a good distribution mechanism for updates to the signature database must be developed. For malware and Anti-virus software, this problem does not exist as the computer that runs the software can usually be used to obtain updates and because the software is in constant use. Live encryption detection, however, is an analysis that is usually performed from an external

medium rather than being installed on the computer and it must be ensured that no or minimal changes are made to the computer to be analyzed so that the update can usually not be performed using the target machine. This should be accompanied by regular health-checks to determine whether or not attempts are made by the target computer to alter the analysis tool.

Bibliography

[1] B. D. Carrier, "Risks of Live Digital Forensic Analysis," *Communications of the ACM,* vol. 49, no. 2, pp. 56-61, 2006.

[2] Guidance Software, "EnCase Forensic - The industry-standard computer forensic solution. Fast, powerful, and proven in courts," [Online]. Available: http://www.guidancesoftware.com/encase-forensic.htm. [Accessed 24 July 2012].

[3] B. Carrier, File System Forensic Analysis, Upper Saddle River, NJ: Addison-Wesley, 2005.

[4] C. L. T. Brown, "Detecting & Collecting Whole Disk Encryption Media," Technology Pathways, LLC, Coronado, CA, 2005.

[5] C. P. Pfleeger and S. L. Pfleeger, Security in Computing, 4th Edition, Upper Saddle River, NJ: Addison-Wesley, 2006.

[6] R. Belfield, The Six Unsolved Ciphers: Inside the Mysterious Codes That Have Confounded the World's Greatest Cryptographers, Berkeley, CA: Ulysses Press, 2007.

[7] U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, "DATA ENCRYPTION STANDARD (DES)," *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION 46-3,* pp. 1-26, 25 October 1999.

[8] National Institute of Standards and Technology, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," *Federal Information Processing Standards Publication 197*, pp. 1-51, 26 November 2001.

[9] National Security Agency, "NSA Suite B Cryptography," 8 June 2012. [Online]. Available: http://www.nsa.gov/ia/programs/suiteb_cryptography/. [Accessed 25 July 2012].

[10] R. Anderson, E. Biham and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," Cambridge, England, 1998.

[11] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall and N. Ferguson, "Twofish: A 128-bit Block Cipher," Minneapolis, MN, 1998.

[12] "TRUECRYPT OPEN-SOURCE ON-THE-FLY ENCRYPTION," [Online]. Available: http://www.truecrypt.org/docs. [Accessed 25 July 2012].

[13] J. Leyden, "Brazilian banker's crypto baffles FBI," The Register, 28 June 2010. [Online]. Available: http://www.theregister.co.uk/2010/06/28/brazil_banker_crypto_lock_out/. [Accessed 25 July 2012].

[14] JADsoftware Inc., "Encrypted Disk Detector," JADsoftware, [Online]. Available: http://www.jadsoftware.com/encrypted-disk-detector. [Accessed 17 July 2012].

[15] U.S. Department of Justice, "NIJ Special Report - Electronic Crime Scene Investigation: A Guide for First Responders, Second Edition," U.S. Department of Justice, Office of Justice Programs, Washington, DC, 2008.

[16] K. Mandia and C. Prosise, Incident Response - Investigating Computer Crime (1st Ed.), New York, NY: Osborne/McGraw-Hill, 2001.

[17] D. Brezinski and T. Killalea, "RFC 3227: Guidelines for Evidence Collection and Archiving," Internet Engineering Task Force (IETF), 2002.

[18] D. Farmer and W. Venema, Forensic Discovery, Upper Saddle River, NJ: Addison-Wesley, 2007.

[19] U.S. Department of Justice, "NIJ Special Report - Forensic Examination of Digital Evidence: A Guide for Law Enforcement," U.S. Department of Justice - Office of Justice Programs, Washington, DC, 2004.

[20] U.S. Department of Justice, "NIJ Guide - Electronic Crime Scene Investigation - A Guide fpr First Responders," U.S. Department of Justice, Office of Justice Programs, Washington, DC, 2001.

[21] J. Garrett, "Overcoming Reasonable Doubt in Computer Forensic Analysis," SANS Institute, 2006.

[22] E. M. Chan, "A Framework for Live Forensics," University of Illinois at Urbana-Champaign, Urbana, IL, 2011.

[23] R. Battistoni, A. Di Biagio, R. Di Pietro, M. Formica and L. V. Mancini, "A Live Digital Forensic system for Windows networks," *IFIP International Federation for Information Processing, Processdings of the IFIP TC 11 23rd International Information Security Conference*, vol. 278, pp. 653-667, 2008.

[24] M. A. Caloyannides, N. Memon and W. Venema, "Digital Forensics," *IEEE Security & Privacy,* no. March/April 2009, pp. 16-17, 2009.

[25] B. Hay, K. Nance and M. Bishop, "Live Analysis - Progress and Challenges," *IEEE Security & Privacy,* no. March/April 2009, pp. 30-36, 2009.

[26] F. Adelstein, "Live Forensics: Diagnosing Your System Without Killing It First," *Communications of the ACM,* vol. 49, no. 2, pp. 63-66, 2006.

[27] K. J. Jones, R. Bejtlich and C. W. Rose, Real Digital Forensics, Upper Saddle River, NJ: Addison-Wesley, 2005.

[28] H. M. Jarrett, M. W. Bailie, E. Hagan and S. Eltringham, Prosecuting Computer Crimes, U.S. Department of Justice - Office of Legal Education, Executive Office for United States Attorneys, 2010.

[29] T. Menzies, A. Dekhtyar, J. Distefano and J. Greenwald, "Problems with Precision," 2007.