COUNTERING MALICIOUS DOCUMENTS AND ADVERSARIAL LEARNING

by

Charles Smutz
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

_____	Dr. Angelos Stavrou, Dissertation Director
_____	Dr. Daniel Barbará, Committee Member
_____	Dr. Daniel B. Carr, Committee Member
_____	Dr. Duminda Wijesekera, Committee Member
_____	Dr. Stephen G. Nash, Senior Associate Dean
_____	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering

Date: _____ Summer Semester 2016
George Mason University
Fairfax, VA

Countering Malicious Documents and Adversarial Learning

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Charles Smutz
Master of Science
George Mason University, 2009
Bachelor of Science
Brigham Young University, 2006

Director: Dr. Angelos Stavrou, Professor
Department of Information Technology

Summer Semester 2016
George Mason University
Fairfax, VA

Acknowledgments

I thank my dissertation committee members for crucial guidance, feedback, and patience as I've conducted this research. I also acknowledge the feedback received from numerous reviewers of this work. I recognize the support of my employer, Lockheed Martin Corporation, as I've conducted this research.

Table of Contents

List of Tables

List of Figures

Abstract

COUNTERING MALICIOUS DOCUMENTS AND ADVERSARIAL LEARNING

Charles Smutz, PhD

George Mason University, 2016

Dissertation Director: Dr. Angelos Stavrou

In order to exploit the large number of vulnerabilities offered by user applications, malware is often distributed through insertion in document, media, and application files. This avenue for malware propagation is also frequently enabled by user response to a ruse, such as opening a fake invoice or following a link to a purported shipping tracking site. Embedded malware is used in a wide range of criminally motivated attacks, but it is also used in highly targeted espionage attacks. Targeted attacks are particularly challenging due to the increased resources of the attacker, which make novel malware, very specific social engineering, and persistent exploitation attempts feasible.

Current approaches to defeating embedded malware include exploit protections such as data execution prevention (DEP) and address space layout randomization (ASLR), which are implemented in most current operating systems. Signature matching systems, such as antivirus tools, are deployed ubiquitously to prevent malware propagation. Other approaches to embedded malware detection include dynamic analysis and machine learning based methods. While these defense mechanisms make malware distribution more difficult, common file formats remain widely used carriers for malware. In this thesis, I build on existing work to advance both exploit protection and malware detection. While the methods I introduce apply to embedded malware generally, I focus my evaluations on document file

formats, such as Adobe PDF and Microsoft Office, because these file formats have been used the most in targeted attacks [20, 49].

To improve exploit protection, I propose modifications to document files that introduce exploit breaking entropy into the reader application without changing the user visible content. This method is called document content randomization (DCR). DCR functions by randomizing the layout or encoding of document content which changes the raw document file and document reader memory. DCR is effective in preventing many current exploits, but can be circumvented by scripting and use of external content.

Recognizing that exploit protections can be circumvented, I also advance detection of malicious documents. To complement other detection mechanisms, I propose use of a Random Forest based classifier relying on features derived from document metadata and structure. This detector, called PDFrate, provides high malware detection rates, even when operating on previously unseen malware samples. PDFrate is available to researchers through an online service. Due to the high classification rates and availability of PDFrate, it has been subject to numerous recently published evasion studies [23, 70, 71, 107, 120].

Adversarial learning can limit the effectiveness of machine learning based classifiers through training set poisoning or mimicry attacks that operate by subverting feature extractors or using knowledge of trained models to create evasive samples. Since preventing all forms of evasion is not feasible, I propose mechanisms that detect classifier degradation due to novel malware or mimicry attacks. Ensemble classifier mutual agreement analysis measures the coherence of votes in an ensemble to determine when the classifier is providing an accurate prediction. If the individual classifiers do not agree on the prediction, the prediction is not trusted and must be found through another source. Mutual agreement analysis is shown to be effective against most contemporary mimicry attacks and assists in optimizing classifier retraining. While the effectiveness of mutual agreement analysis is dependent upon the accuracy of the underlying classifier and susceptibility of feature extractor to subversion, it raises the bar for mimicry attacks.

Chapter 1: Introduction

1.1 Background

Computer misuse includes unauthorized access to computer systems, compromise of user privacy, exposure of confidential information, distribution of unwanted messages (SPAM), and denial of service. While many attack vectors are used, distribution of malicious software (malware) remains a key enabler of computer system abuse. In years past, direct exploitation of the operating system through internet worms such as Code-Red [78], Blaster [10], and Witty [99] was common. However, as use of firewalls has expanded, operating system security has improved, and automated updates are used widely, attackers have focused on vulnerabilities in applications.

Today, most exploits target vulnerabilities in client applications such as document readers, multimedia programs, and internet browsers. For example, in 2008, Microsoft reported that the number of operating system vulnerabilities had decreased steadily over the past five years and that vulnerabilities in applications constituted over 90% of disclosed vulnerabilities [47]. Hence, large scale attacks such as the Storm Worm [51] evolved to use email attachments or links as the propagation method. More recently, exploit kits, such as Blackhole or Incognito [41, 46] are commonly used to automate exploitation of browsers and browser plugins by using one of many available exploits depending upon the software exposed by the victim system.

As malware propagation has shifted from attacking network services to user facing applications, exploitation often relies on user action such as opening an email attachment or following a link. Indeed, social engineering, or tricking the user to take an action that leads to exploitation of the user's system or exposure of sensitive information is common [87]. Use of specific file formats, such as documents attached to emails, often helps make the ruse

seem more realistic. For example, an attacker may use the promise of an attached purchase order or a link to a shopping promotion to entice users to open a malware laden document or follow a link to malware. On the other hand, some infection occurs without interaction of the user through drive-by downloads of malware [28, 37, 109]. In either case, embedding malware in media files or web content also helps evade detection. Most file formats allow for various forms of encoding, linking, embedding, and scripting that can be used to evade signature matching and advance exploits [32, 38, 89].

Malware is used for various nefarious purposes. Botnets, or large networks of compromised systems, are used to assist in further malware propagation or to wage distributed denial-of-service attacks [5, 113, 124]. Financial fraud has long been one of the top goals of malware [50, 79]. Ransomware, such as CryptoWall, which performs extortion by promising decryption of maliciously encrypted user files, is now a prevalent problem [40, 45]. Some malware is used in targeted attacks to conduct espionage. Beyond traditional national security and diplomatic goals, these attacks also seek to undermine the economic competitive advantage of private companies and disrupt the activities of non-governmental organizations (NGOs) [2, 6, 14, 20, 49, 65, 115]. Reports that these attacks are performed on the behalf of governments are especially concerning because of the resources that can be expended in these attacks [3, 31, 39, 42, 54, 73, 75, 94]. These targeted attacks typically involve emails with very specific information related to the intended victim and malware embedded in Adobe PDF or Microsoft Office documents [1, 20, 49, 65].

Validating digital signatures or using vetted application stores are both common approaches to preventing malware execution, but these mechanisms are often circumvented or subverted [123]. Operating system level protection mechanisms such as address space layout randomization (ASLR) [86] and executable space protection [33] are used widely, but these mechanisms are commonly defeated [111]. Preventing software flaws and automatically updating software also limit exploitation, but preventing all software exploitations is not practical.

Another means of preventing the spread of malware is through detection. Antivirus

and intrusion detection systems (IDS) [91] are deployed widely, using signatures to detect known malware. These systems are limited in the their ability to stop previously unseen malware variants so they are often complemented with dynamic analysis [35]. Dynamic analysis is used in limited situations due to difficulty in modeling end systems (and users), computational expense, and need for an expert to interpret software behavior. In order to improve malware detection systems using both statically and dynamically extracted features, machine learning is employed [9, 30, 100, 106]. Machine learning based classifiers promise to provide better detection of new malware variants, but they are susceptible to mimicry attacks.

Adversarial learning concerns learning based systems that operate in environments where attackers actively seek to avoid detection, such as malware detectors or SPAM filters [52]. Mimicry attacks leverage knowledge of features and the learned model to create attack samples that appear benign to the classifier. Many recent studies have shown that mimicry attacks against machine learning based classifiers are practical [71,107,120]. Other studies have demonstrated the importance of keeping training sets free from influence of attackers [11, 29, 63, 80]. Improving evasion resistance of learning based malware detectors remains an open problem.

1.2 Thesis Contributions

I seek to prevent the use of common file formats as a carrier for malware. I propose interrupting exploits through entropy inducing modifications to documents that do not change document rendering [101]. Because it is not practical to block all exploits, I also advance methods for malware detection. I propose a machine learning based classifier that relies on features derived from document structure and metadata [100]. This learning based detector, PDFrate, has been the target of numerous recent adversarial learning studies [23, 70, 71, 107, 120]. In order to counter mimicry attacks, I devise a novel method of detecting evasion attacks using ensemble classifier introspection [102].

The key contributions are:

Document Content Randomization An exploit protection technique using permutations to documents to introduce entropy in the reader application

PDFrate A malware detector for PDF documents using metadata and structural features with a Random Forests classifier

Mutual Agreement Analysis A method of detecting failures in ensemble classifiers by measuring the level of coherence in voting

1.3 Thesis Outline

Following this introductory chapter, Chapter 2 contains an overview of related work. I demonstrate how exploits in Office documents can be defeated through document content randomization in Chapter 3. As a complementary approach, Chapter 4 describes classification of malicious PDF documents employing structural and metadata features. My approach to countering recent mimicry attacks against PDFrate utilizing ensemble classifier mutual agreement analysis is found in Chapter 5. In addition to numeric features, my study to determine the degree to which term based features improves PDF classification is reported in Chapter 6. In Chapter 7, I study machine learning based detection for file formats beyond PDF documents.

Chapter 2: Related Work

This thesis builds upon years of research in the areas of exploit protections, malware detection, and machine learning techniques. I seek to defeat all forms of malware, but am especially motivated by targeted attacks in which threat actors aggressively pursue specific organizations in order to achieve strategic objectives such as espionage.

Computer systems are exploited for a large number of end goals including financial gain, political demonstration, and espionage. Network intrusion attempts for the purpose of espionage are often called targeted attacks [2, 6, 73, 115]. These targeted attacks differ from financially motivated attacks in both methods and consequences. Herley [50] argues that targeted attacks are not effective for an attacker driven solely by economic gain. Li et al. expose malware obtained from two email based attacks in order to prove that target persistent attacks exist [65]. Hardy et al. find that espionage focused attacks involve varying levels of technical sophistication but usually involve a high degree of social engineering [49]. Blond et al. also perform analysis of targeted attacks against members of a non-government organization [20]. These studies confirm that while other exploitation vectors are studied, emails with malicious documents as attachments remain an extremely common methodology for targeted attacks. This thesis seeks to detect malware from all threat types, but the serious consequences of targeted attacks provide additional impetus and focus. I seek to improve identification of malicious documents, a common exploit vector used by targeted attackers. I build upon previous in characterizing targeted espionage threats by demonstrating that I can train a high accuracy classifier which will separate targeted attacks from opportunistic threats.

Amin [7] advances a machine learning based approach for detecting targeted malicious email. This thesis is related in that it seeks address delivery of targeted attacks and uses a Random Forest classifier. While I focus on malware-centric features of documents, Amin

finds that features of emails reflecting persistent attackers and recipient targeting can improve detection rates. Recognizing that malware construction utilities are often re-used, Donaldson studies fingerprinting of specific document generation tools using structural elements of documents [34]. I also study the repetition of specific structural and metadata items in malware samples, knowing that these items often coincide with, but are not necessary to, the existence of malware.

Seeking to defeat all threat types, there is a large corpus of research in probabilistic exploit mitigations that are typically implemented in the operating system of computing systems. Address space layout randomization (ASLR) [86] is adopted widely. ASLR is effective in defeating many classes of exploits, but is circumvented through limitations in implementation [97], use of heap sprays [19, 117], or data leakage [95]. As return oriented programming and similar techniques [96,118] have become popular, mechanisms to relocate or otherwise mitigate code (gadget) reuse have been proposed [59,84,122]. ALSR incurs little run time overhead because it relies on virtual memory techniques where address translation and relocation are already performed.

Code level approaches such as instruction set randomization have also been proposed but are prohibitively expensive [60]. Data space randomization enciphers program data with random keys, but this method is not feasible in practice due to deployment difficulty and computational expense [16]. My approach focuses on misuse of data in exploits but differs in that I propose modifications to the malware hosting document to induce computationally efficient entropy in the document reader. Other policy enforcing techniques, such as executable space protections (DEP or W$\oplus$X) [33], are used widely. Despite the many proposed exploit mitigations, exploits are still practical on modern systems [111].

Because exploit protection does not prevent all malware propagation, detection of malware, including malware embedded in common file formats such as documents, is a current research topic. Pattern matching and malware clustering have been studied extensively [13, 22, 56, 58, 68, 121]. Despite known limitations in detecting new samples, signature matching and policy enforcement techniques are found in commonly deployed systems [91,103]. More specifically, malware bearing documents have been the subject of much research over the years. Rautiainen describes the exploitation techniques and vulnerabilities used in PDF files [90]. Wressnegger et al. disclose that the very obfuscation methods used to evade signature matching in Office documents can be detected using probable plaintext attacks [119], but this approach is limited to situations where weak cryptography is used.

Stolfo et al. perform n-gram analysis on various file types including Office and PDF documents, finding that this analysis provides an improvement over signature based detection [108]. Li et al. also study n-gram analysis in Office files [67], finding that parsing files into individual document objects is necessary to achieve high detection rates. While existing malware can be detected in this manner, byte level statistical analysis is shown to be fragile due to the ability to obfuscate malware and inherent issues in separating malicious code from other data [74,105]. Li et al. also demonstrate that examining dynamic traces of documents opened in sandbox systems can detect many forms of malware, but challenges with this approach include separating benign from malicious content and ensuring complete execution of stealthy malware. Tabish et al. layer various statistical measures of divergence or entropy on top of standard n-gram data to achieve modest classification rates on various file types including documents [112]. While I do not rely on content based features, I do study the degree to which the specific values observed in structure and metadata can be used to improve classification rates.

Cova et al. perform execution of javascript and utilize features derived from both system behavior and inspection of generated content in a Bayesian classifier to identify malicious javascript [28]. Although originally designed for web based javascript, this functionality

has been extended to detect malicious content in PDF files[1]. Laskov and Šrndić parse and extract lexical features from javascript embedded in PDFs to provide classification using Support Vector Machines (SVM) [64]. Tzermias et al. combine basic structural analysis with emulation of the Adobe reader javascript API in a PDF to detect shellcode obfuscated by javascript [114]. Maiorca et al. perform classification based on javascript API references. Liu et al. inject monitoring javascript into documents, combining dynamic monitoring with static features at run time [69]. I also study document based malware execution, but instead of focusing on detection, I seek to defeat exploits through document content randomization [101].

Cross and Munson [30] use an instrumented reader application and dynamic analysis to extract structural features of PDF documents to be used in a machine learning based classifier. Maiorca et al. advance a pattern matching based approach focusing on PDF document structure and utilizing a Random Forest based classifier [72]. These studies are very similar in concept to my work. My thesis differs in that I use a more comprehensive feature set employing both metadata and features derived from multiple structural markers [100]. Šrndić and Laskov further advance static analysis of PDF document structure by employing more complete parsing of documents to model the hierarchical structure of documents utilizing a linear SVM classifier [106]. Maiorca et al. advance the combination of structural features with scanning of content for suspicious conditions including malformed objects [70]. Despite these varied techniques for malware detection, malware propagation still occurs due to evasion through polymorphism and mimicry attacks.

Beyond detection of malicious documents, machine learning techniques are used pervasively to solve computer security problems such as detection of unsolicited email [15, 57, 92], malicious downloads [28, 88], detection of account misuse in social networks [36, 110], and detection of malware in other file formats such as Java Archives [93] and Android applications [9]. Due to the pervasive use of machine learning and statistical techniques, adversarial learning, or the use of machine learning in a hostile environment, is a popular

[1] https://wepawet.iseclab.org/

research topic. Huang et al. provide a taxonomy and models for describing the threats that machine learning based classifiers face [52].

Some studies have proposed methods for creating effective classifier based intrusion detection systems [12, 43, 104]. Many studies have addressed the importance of data sanitization or adversarial influence at training time [11, 29, 63, 80]. Yet other adversarial learning studies focus on evasion of the deployed classifier [17, 18]. Maiorca et al. study a reverse mimicry attack against detection systems including PDFrate [71]. This attack method seeks to minimize the visibility of the malicious objects in PDF documents. Šrndić and Laskov perform a systematic evaluation of mimicry attacks against PDFrate, modeling an attacker that has knowledge of PDFrate's features, training set, and classifier [107]. Xu et al. use a genetic programming approach employing multiple randomized mutations to attack PDFrate [120]. Another current research thread demonstrates that many malware detectors are defeated by exploiting weaknesses in their parsing routines [48, 55]. Carmony et al. demonstrate that most PDF malware detectors can be foiled by exploiting bugs and limitations in PDF parsers [23]. I also focus on evasion of a classifier during operation, but instead of focusing on strategies for evasion, I propose a means of detecting these evasion attempts.

Recent work has demonstrated that the diversity in ensemble classifiers can improve malware detection rates [62, 76, 116, 121]. Few studies, however, advance practical strategies for detection of evasion attempts against these ensemble classifiers. Chinvale et al. proposed the use of mutual agreement between a small number of independent SPAM filters to optimize individual classifier re-training necessary due to drift in input data [26]. I extend this approach to introspection of ensemble classifiers in order to provide a per observation confidence estimate at test time [102]. My thesis differs fundamentally from Chinvale et al. in that they use the majority result of their ensembles as ground truth for re-training of individual classifiers while I focus on identifying the specific examples where the ensemble prediction is not trustworthy. In short, Chinvale et al. use diversity in ensembles to improve classifier performance. I use diversity to identify when resorting to external ground truth

is necessary. Nissim et al. utilize inspection of SVM margins to identify samples to be added to the training set, but they do not address mimicry attacks [82]. I study the factors that enable diversity based confidence estimates in ensembles using variations of bagging. Going beyond natural drift or novel attacks, I apply mutual agreement analysis to focused mimicry attacks.

Estimation of confidence based on knowledge of a population has long been foundational to statistical methods [81]. Contemporary research has demonstrated how these confidence estimates can be applied to a machine learning based classifier deployed in an online setting [98]. However, since these approaches rely on new observations matching the distributions of training samples for which ground truth is known, they are not applicable to intrusion detection systems that face novel observations and mimicry attacks. Rather than seeking to quantify the overall accuracy of a classifier, I identify the individual observations for which a classifier cannot provide a reliable response. My approach makes use of data already provided by the classifier, without additional appeal to ground truth or independent outlier analysis.

Chapter 3: Content Randomization in Office Documents

I studied methods to prevent exploits in common file formats through modifications to the input data. Due to the popularity of document based exploits, I sought to defeat exploits through document content randomization (DCR). The goal of DCR is to make modifications to documents that introduce exploit disrupting changes without interfering with normal document use. This approach is inspired by existing execution environment based exploit mitigations such as ASLR and data randomization, but is implemented in the document itself.

I designed and evaluated exploit protections using transformations performed on documents between production and consumption. Document content fragment and encoding randomization are effective in scrambling exploit critical content in document files and in document reader process memory. I evaluated the ability to mitigate current exploits in Office 2003 (.doc) and Office 2007 (.docx) file formats using hundreds of malicious documents, demonstrating a memory misuse exploit block rate of over 96%. The overhead of transforming documents is comparable in run time to a common antivirus engine and the added latency of opening a content layout randomized document is negligible for .doc and about 3% for .docx files. The transformed documents are functionally equivalent to the original documents, barring the exploit protections that are induced. The evasion resistance of content randomization is rooted in the number of raw content permutations possible. File content randomization should be applicable to other file formats as complementary controls force attackers to use direct access to file content to advance their attacks.

I also studied exploit protection mechanisms using resource consumption, such as large memory allocations. While this approach can mitigate some exploits, it is not practical for use due the extreme computational resources required.

3.1 Microsoft Office File Formats

There are multiple commonly used Microsoft Office file formats. The OLE Compound Document Format was used as the default by Office 97-2003 and is used in the files whose extension is .doc, .ppt, or .xls. Beginning in Office 2007, the default file format is Office Open XML, which uses the .docx, .pptx, and .xlsx extensions.

3.1.1 OLE Compound Document Format

The file format used by Office 97-2003 is called by many names including Compound File Binary Format [77,83] and OLE Compound Document Format. I refer to this format as the "OLE" file format throughout this thesis, as many of the libraries and utilities for parsing this format use variations of this name.

The OLE Compound Document Format supports the storage of many independent data streams and borrows many structures from filesystems, especially the File Allocation Table (FAT) filesystem. OLE files are used as a container for many file types, including Office 97-2003 (.doc/.ppt/.xls), Outlook Message (.msg), Windows Installer (.msi), Windows Thumbnails (Thumbs.db), and ActiveX or OCX controls (.ocx). All of these file formats use the OLE format as a base container, implementing their own data structures inside of streams stored by the OLE format. In this way, the OLE file format can be compared to the zip archive that serves at the base container for diverse file formats including Java Archives (.jar), Office Open XML Documents (.docx/.pptx./.xlsx), and Mozilla Extensions (.xpi).

Like a filesystem, the OLE format is comprised of many data blocks or sectors. The vast majority of OLE files use a 512 byte sector size, but other sizes are possible. The first sector of an OLE file is a header which contains the file signature 0xd0cf11e0, various parameters for the file such as the size of the blocks, and the offsets of key data structures. This could be compared to the super block of a filesystem.

The first data block also contains the Master Sector Allocation Table, which contains pointers to the sectors used by the Sector Allocation Table. The Sector Allocation Table is a table of pointers to the next sector in each data stream. Following the chain of pointers

to data sectors in this allocation table allows the individual data streams to be constructed from potentially arbitrarily ordered sectors. The majority of the data blocks in a typical OLE file are allocated to the storage of the embedded file streams. Some data sectors in a file may not be used so it is possible to have OLE files larger than the logical contents of the file.

The individual data streams are logically organized in a hierarchical structure similar to the file/folder organization of a filesystem. There is a directory entry for each OLE stream containing the name of the file, the location of the first sector, and other metadata such as modification times. These entries create a tree data structure starting at the root entry.

The OLE format also supports small or mini data streams. These are similar to the normal data streams, but are intended to be much smaller to prevent the space wasted by allocating a full sector to a very small data stream. Typically, the block size for these streams is 64 bytes. Another sector allocation table and these smaller sectors are embedded into normal sector size data streams.

Figure 3.1 shows the layout of a typical OLE Compound Document Format file. Except for the header, which must be located in the first sector, all of the contents in an OLE file can be located arbitrarily within the OLE file. Furthermore, there is no requirement for the various data streams to be arranged in order or in contiguous blocks, although it is the norm.

The OLE format serves as a container for arbitrary data streams. Individual file formats use this basic container but implement their own format for the data in the various streams. I do not describe the particular OLE-based file formats because they vary widely, they are often proprietary and poorly documented, and my study relies on the container capability of OLE files.

3.1.2 Office Open XML File Format

The Office Open XML (OOXML) file format became the default file format for Office documents starting in Office 2007. These files use the extensions of .docx, .pptx, and .xlsx.

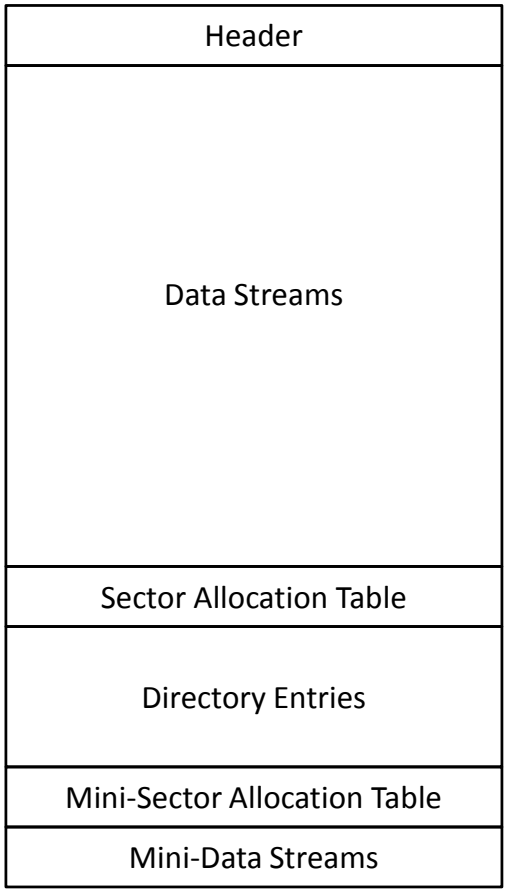


Figure 3.1: OLE Compound Document Format

This format has been codified in international standards ECMA-376[1] and ISO/IEC 29500[2].

The OOXML file format uses a zip file as a container, with individual objects stored as files in the zip. The majority of the content in an OOXML file is XML data. The document content is represented as XML with markup that is unique to the OOXML format, but which is generally similar to other markup such as HTML. The contents of an OOXML document can be modified by unzipping the archive, modifying it with a text editor, and zipping the archive. However, the relatively complex markup requires extensive knowledge

[1]http://www.ecma-international.org/publications/standards/Ecma-376.htm
[2]http://www.iso.org/iso/catalogue_detail?csnumber=51463

to make major changes.

While text and formating can be represented using XML, other content, such as images and other documents, are embedded in the document as separate files in the zip archive. Some of the binary objects embedded in OOXML files use the OLE format. Examples of these files include Office 2003 files, some multimedia files, equations, ActiveX controls, and executables.

Throughout this thesis, I refer to the container format common to Office 2003 and many other files as OLE. I refer specifically to Office 2003 and Office 2007 files by their extension: .doc and.docx respectively. While I refer to these file formats by the file extension of the document files (.doc/.docx) for brevity, I also included the presentation (.ppt/.pptx) and spreadsheet (.xls/.xlsx) files in my evaluations. My study relies on file format characteristics that are common across the document, presentation, and spreadsheet file variations.

3.2 Microsoft Office Exploit Protections

I briefly describe some of the most important exploit protections provided by Microsoft Office. Macro based viruses have long been an issue in Office documents. All recent Office versions disable the automatic execution of macros. The OOXML file format assigns macro based files a separate extension, such as .docm instead of .docx, making inadvertent execution of macros extremely difficult. The OOXML format is designed to minimize the amount of binary content and improve the readability in the text content of documents, making the file format more easy to validate and simplifying the parser.

Data Execution Prevention (DEP) was enabled in Office 2010, which prevents execution of arbitrary shellcode in the heap and has generally forced the adoption of return-oriented programming (ROP) code reuse techniques. ASLR was enabled by default in Windows Vista. Office running on versions of Windows since Vista will have the benefit of ASLR for operating system libraries, but ASLR was not enabled for all Office provided libraries until Office 2013. Hence, up until Office 2013, one observes use of ROP gadgets from Office libraries without the need for an ASLR bypass.

3.3 Approach

Inspired by the simplicity and generality of ASLR-like techniques, I seek to obtain similar exploit mitigation outcomes through transformations to input data. The properties of a document can often directly and predictably influence various run time attributes of the opening application. The memory of a reader program is necessarily influenced by the file that it opens. I seek to find practical ways to make exploitation more difficult by content induced variations to the document file and reader memory. I call this general approach Document Content Randomization (DCR).

I propose two specific forms of document entropy infusion: document content fragment randomization (DCFR) and document content encoding randomization (DCER). DCFR is analogous to ASLR in that the order of data blocks in the document file is randomized. DCER can be compared to data space randomization because I randomize file level encoding. Both of these approaches apply to data stored in document files, but they can affect memory as files and subfiles are loaded into memory.

These transformations are envisioned to operate on documents during transfer between the potentially malicious source and the intended victim. They could be employed in network gateways such as email relays or web proxies where modification is already supported. In practice, filtering based on blacklists and antivirus scanning is already common at these points. The modifications to the document could also be implemented on the client. For example, the web browser could employ the mechanisms presented here at download time, similar to other defenses such as blacklists and antivirus.

I focus on Office documents, but most of the high level principles discussed here apply to other file formats. Specifically, I seek to mitigate exploits in two common document formats: Office 2003 (.doc) and Office 2007 (.docx).

3.3.1 Content Randomization in .doc Files

The most promising opportunity to apply DCR to .doc files is at the raw document file level. Malicious content is often stored in the raw document file and accessed through the

filesystem during exploitation. Typically, file level access of malicious content occurs later in the exploitation phase and this content is usually malicious code, whether it be shellcode or a portable executable. Sourcing malicious content from the file is surprisingly common in document based exploits.

I observed obfuscated portable executables embedded in the raw document file of 96% of the malicious Office 2003 documents in the Contagio document corpus [85]. This set of malicious documents, observed in targeted attacks, includes files that were 0-day attacks when collected. Retrieving additional malicious content from the raw document file is also common in PDF files used in targeted email attacks, while web based PDF exploits usually load their final payload through web download.

File level access is most frequently achieved through standard file access mechanisms, such as reading the file handle. Because most client object exploits, including document exploits, are triggered by opening a malicious file, a handle to the exploit file is usually already available in the reader application. While the malicious content may be embedded raw into the document file, exploits typically employ signature matching evasion techniques, such as trivial XOR encryption. The malicious content accessed through the raw document is sometimes accessed by offset, but typically an egg hunt is employed, where the file is searched for a specific marker. In most .doc files, I observed the shellcode and portable executables embedded within the bounds of the structure of the document, but simply appending malicious content to the end of an existing document is possible and is used sometimes.

I defeat raw file reflection by malware by performing file level content fragmentation (DCFR). OLE based file formats such as .doc files are especially accommodating of this technique. Typically, the streams in an OLE file are sequentially stored. However, reordering can occur and is expressly allowed. To implement this approach I built an OLE file block randomizer. It simply creates a new OLE file functionally equivalent to the original except that the layout of the data blocks is randomized. This is accomplished by randomizing the location of the data blocks, and then adjusting the sector allocation tables and

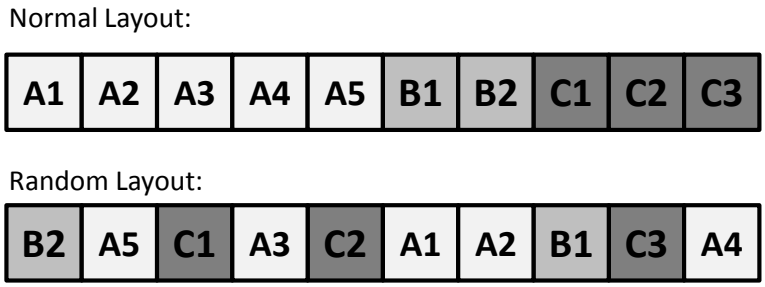


Figure 3.2: OLE Fragmentation: The Order of Blocks in Data streams is Randomized, Fragmenting the Payloads

directory data structures accordingly. This is essentially the inverse of running a filesystem defragmentation utility. An example of fragmentation of three OLE data streams is given in Figure 3.2. Note that the blocks in the data streams are typically sequentially arranged. I randomize the order of the data blocks across all the streams. In the event that any data exists in the raw document file stream but is not contained in valid OLE sectors, the data is not transfered to the new randomized document.

Reordering data blocks, or DCFR, in an OLE file provides a consistently effective and quantifiable way to prevent access to malicious content in raw document files without impacting normal use. Since OLE files do not implement any form of encoding at the container level, DCER is not a practical option.

3.3.2 Content Randomization in .docx Files

I also studied the use of document content in .docx exploits. I found a small number of OOXML files where the raw zip container was accessed for a malicious payload. These attacks simply included the malicious payload, usually an encrypted portable executable, in the zip file without compression. It is then trivially located in the file through an egg hunt, similar to that done in OLE files.

I devised two simple ways to introduce entropy in the OOXML file. First, I randomized the order of the files in the zip archive. This defeats access based on offset. I also recompressed the zip data streams, randomly selecting one of four deflate compression levels

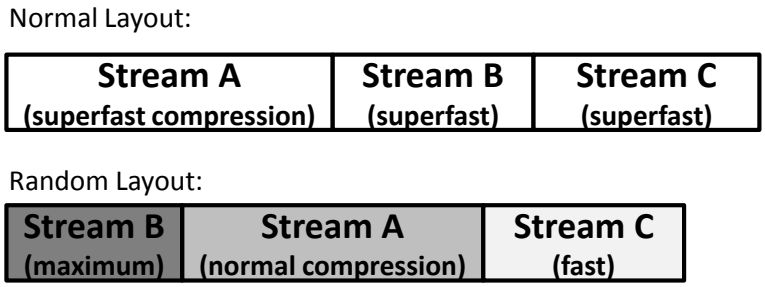


Figure 3.3: ZIP Encoding Randomization: The Order and Compression Level of Data Streams is Randomized

(superfast, fast, normal, maximum). Figure 3.3 demonstrates transformation of a simple zip file with three subfiles. Note that the order of the files in archive and the compression used on each file is randomized. Office uses superfast compression, the lowest compression level, so my archive randomization usually results in smaller files. Compression level randomization is enough to foil simple access to file content, even if an egg hunt is used. Therefore, content encoding randomization (DCER) applied at the file level is applicable to some .docx exploits.

However, most .docx exploits gain access to the final malware payload through a web download or through an egg hunt in memory. I found a common method of performing scriptless heap sprays in contemporaneous exploits that can be mitigated by DCFR [4, 66]. In this heap spray technique, first observed in CVE-2013-3906, many ActiveX objects containing primarily heap spray data are read when the document is opened and loaded into the heap. These objects are loaded into memory raw, without interpretation or parsing. It is not clear why these objects are loaded into memory in this manner, while other embedded files do not receive the same treatment. Dynamic analysis by the author confirmed that these embedded ActiveX objects are loaded directly into memory, while most other data from the document is not loaded into memory wholesale. Even if these ActiveX controls are not activated, they represent a simple and effective way to introduce content directly into the memory of the reader program.

Heap sprays are used to defeat ASLR. They ensure that the malicious payload can be

located with high certainty through duplication of the malicious payload across a large memory address range, even if the address of a single copy cannot be predicted. Only one copy of the malicious payload is needed for successful exploitation. Traditionally, heap sprays contain shellcode. However, DEP prevents execution from the heap. In the case of exploits targeting systems with DEP, the heap is commonly sprayed with ROP gadgets and a stack pivot is used to move the stack into the sprayed region. These attacks successfully evade ASLR and DEP. I observed this technique for scriptless heap sprays used for both traditional shellcode and to implement fake stacks containing malicious ROP chains. While these two techniques have been observed, this ability to easily and predictably influence the reader process's memory could be used for other attacks such as object corruption exploits. This general technique is also used to load single copies of arbitrary content, including portable executables, into memory that is later egg-hunted and used in exploits.

Since these ActiveX objects use the same OLE container format that Office 2003 documents use, I use the same OLE fragmentation techniques to defeat these scriptless heap sprays. I randomized the layout of all OLE files embedded in .docx files, regardless of their role. When these objects are loaded into RAM, the content is scrambled, but can still be retrieved by a document reader that implements the OLE decoding routines. These scriptless heap sprays in .docx files represent an example of how document content directly influences reader memory.

For .docx files, I perform both file level encoding randomization and fragmentation of objects to be loaded into memory.

3.3.3 Strength of Content Randomization Mechanisms

Like other probabilistic exploit protections, one can calculate the likelihood of exploit success in the face of brute force attacks against DCR. Methods such as ASLR obfuscate the location of malicious payloads. Document content randomization does this as well. However, content based malicious payloads are very frequently located via egg hunts or are duplicated in heap sprays, obviating randomized relocation. In practice, the primary protection

power lies in randomization of the content representation, whether through fragmentation or through encoding.

In the simple case, the probability of a randomly fragmented payload being in proper order is the inverse of the number of possible permutations or $1/n!$ where n is the number of fragments. In practice, this should be adjusted to account for other data mixed in with the malicious payload, repetition of the malicious payloads, and other limitations or constraints. For example, when OLE DCFR is employed, the number of fragments that influence the possible permutations is not just the number of fragments in the malicious payload, but includes all of the sectors that are randomized.

When I perform DCER on .docx files, I randomly select among four deflate compression levels. This is adequate for all the samples we observed where DCER has affect because they all involve data streams that are originally uncompressed. This is, however, a very small number of permutations. Part of the strength of DCER is also rooted in how difficult the encoding of the content is to reverse or circumvent. Compression makes generating a specific post compression malicious payload more difficult through transformations and restrictions in the encoded output. For example, repeated byte sequences, such as the high order bytes in addresses used in ROP gadgets, are not found in compressed output. Unlike straightforward fragmentation, the constraining power of encoding is more difficult to quantify. Individual implementations of deflate are deterministic, but they are also allowed great latitude in how the encoding occurs. The same data stream can have many byte level representations using the same encoding method. If an entropy inducing compressor/encoder is used, the number of encoding induced permutations could be quantified.

The strength of DCR lies in the ability to fragment or encode malicious payloads in an unpredictable and constraining manner. This strength can be quantified as proportional to the number of randomized content permutations. I address possible DCR evasion approaches in Section 3.6.

3.4 Exploit Protection Evaluation

I evaluated the effectiveness of my content based exploit protections on hundreds of malicious Office documents sourced from VirusTotal[3]. These documents were downloaded daily from the recent uploads to VirusTotal over the course of months. My downloads were limited primarily by my monthly download limit on VirusTotal. I obtained 64,617 unique .doc files between May 2013 and March 2015 and 32,383 unique .docx files between November 2013 and March 2015, averaging 98 .doc and 66 .docx files per day. Of these collected documents, 40,720 .doc and 2,901 .docx files were labeled by at least one AV engine as malicious in a scan conducted two weeks following initial submission. Of these malicious documents, 1,085 .doc and 578 .docx files were labeled by the antivirus engines as utilizing a known exploit. The majority of the non-exploit malicious documents were identified by the antivirus engines as utilizing macros.

I advance methods to break exploits using mechanisms not applicable to pure social engineering attacks. Therefore, I focused my evaluation solely on maldocs leveraging a software vulnerability. Furthermore, to be able to better explain how my mechanisms applied to specific exploits, I used only those maldocs that were labeled by antivirus engines to use a single exploit. I was left with 962 .doc and 363 .docx files after inconsistent exploit labels were removed. Of these documents, I found all exploits for which I was able to replicate successful exploitation and for which there were at least 20 samples. This resulted in three exploits in .doc files and three exploits in .docx files. Surprisingly, the malicious documents were distributed heavily across a small number of particularly popular exploits. For example, the three top exploits in the .docx file types comprised 306 of the 363 files, with 225 of these samples using the most popular exploit. In the event that I had many samples for a given exploit, I randomly selected a subset achieving a maximum of 100 documents to test and a maximum of 50 viable maldocs per exploit. In total, there were 343 documents tested and 217 documents demonstrating successful malware execution across these six sets.

To test for exploitation, I attempted dynamic execution of the Trojan documents by

[3]https://www.virustotal.com/

opening them in a virtual machine. To achieve successful exploitation, I used various configurations of software including both Windows XP and Windows 7 and Office 2007 and Office 2010. The ROP based exploits required specific versions of the libraries from which they reuse code. Since one of the exploits selected for my testing is in Adobe Flash, I also installed the appropriate version of Flash player. I considered the malware execution successful when malicious code was executed or requested from the network that would have been executed. Successful exploitation occurred in 217 or 63% of the malicious documents I tested. I attribute this relatively low malware success rate to VirusTotal being used by malware authors for testing, sometimes testing unreliable or incomplete exploits. For example, in a few of the successful exploits I observed calc.exe, the malware "hello world", as the final payload. There were a small number of apparent false positives by antivirus software as well.

Taking these successful malicious document based exploits, I applied my document content based mitigations and re-ran the documents. I considered the exploit blocked by DCR when the final malware payload was blocked. I observed the differences in malware execution through both host based and network based instrumentation. In a very small number of cases, DCR was not possible due to the malicious document having defective structure. These failures were considered blocked as well, due to the rudimentary file validation provided by performing content randomization.

Generally, the malicious documents I observed employ a portable executable as the final malicious payload. Most of these executables are extracted from the raw document file, many are downloaded from an external server, and a few are extracted from document reader memory. In many of the Trojan documents, the original document file is overwritten by a benign document, which is opened and presented to the user. Most of the malware immediately beacons to a controller node, but a small minority of the malware performed other actions such as infecting other files on the local system. I observed dropped benign documents and malware that correlate to recent reports of targeted attacks against NGOs [20, 49] as well as more opportunistic crimeware.

Table 3.1: DCR Exploit Protection Evaluation

CVE	File Type	Blocked	Total	Block Rate	Effective Mechanism
2009-3129	.xls	36	36	100%	File Fragmentation
2011-0611	.doc, .xls	29	29	100%	File Fragmentation
2012-0158	.doc, .xls	50	50	100%	File Fragmentation
2012-0158	.pptx, .xlsx	4	10	40%	File Encoding
2013-3906	.docx	42	42	100%	Memory Fragmentation
2014-4114	.ppsx, .docx	2	50	4%	File Validation
All	-	163	217	75.1%	-

When the document based exploit is blocked by DCR, the document reader typically crashes. However, sometimes instead of crashing, the reader enters an infinite loop, presumably performing an egg hunt that is never successful. When a decoy benign document is provided by the malware, it is either never opened due to a failure in malware execution or the benign document is scrambled due to DCR and the attempt to open the document fails because the file is invalid. When DCR interrupts file-level access, shellcode that is attempting to extract a portable executable or additional shellcode from the document file is interrupted. When memory fragmentation is effective, it scrambles either shellcode or ROP chains, preventing exploitation earlier. Table 3.1 contains the high level results of my evaluation.

CVE-2009-3129 is triggered by a malformed spreadsheet that causes a memory corruption error. All of the successful exploits were .xls spreadsheet files. In all of these exploits, the pattern of extracting an encrypted portable executable and benign decoy document is employed. Due to raw access to the document file, all of these exploits were defeated by file level DCFR.

CVE-2011-0611 is actually a vulnerability in Adobe software products, including Flash player, but it is most often observed inside of Office documents. This exploit triggers a type confusion error through a malformed Flash file embedded in the Office document. I was able to observe successful exploitation in both .doc and .xls files. Like the other exploits embedded in OLE based file formats, all of the exploits are defeated by file level DCFR

because the malicious executable and decoy document are extracted from the raw document file.

It is interesting to observe that this exploit in Adobe products was used so heavily in Office files. It is likely that part of the reason this exploit was embedded in Office documents was to leverage the social engineering of email based attacks.

CVE-2012-0158 is caused by malformed ActiveX controls that corrupt system state. While originally reported in RTF documents, my VirusTotal sourced malware contained a large number of 2012-0158 exploits in the OLE container as well. I observed successful exploitation in both .doc and .xls files, which was defeated by file level document fragmentation.

I also observed 2012-0158 in OOXML based files. These .docx based 2012-0158 were much less common than the .doc version, making this set the smallest in my evaluation. I observed both .pptx slideshows and .xlsx spreadsheet files containing viable exploits.

This vulnerability exists in the MSCOMCTL library which handles ActiveX controls. Until May 2014 (CVE-2014-1809), ASLR was not enabled on this library on all versions of Office (including Office 2013) and on all version of Windows (including Windows 7 and Windows 8). Since this library is easily locatable, it is trivial to reuse code from the same library as is used for the initial vulnerability. Due to this lack of OS level exploit mitigations and the simplicity of exploitation, DCR, including memory fragmentation, does not block this exploit. It is noteworthy that since the ActiveX controls used in this exploit are OLE files, my DCR mechanisms fragmented these objects. However, since the access to these objects comes through legitimate means, the layout randomization provides no mitigation power.

However, some exploits are foiled because they use anomalous access to the raw document file. In the case where the raw document is accessed, the encrypted malicious payload is stored in the zip container without compression. My recompression of the zip streams with a randomly selected compression level defeats this file level access.

CVE-2013-3906 is a vulnerability in the TIF image format parser that permits memory

corruption resulting in possible code execution. This exploit was manifest in .docx documents. Some of these exploits use ROP chains, while some use traditional shellcode. The ROP based exploits can evade DEP using a stack pivot and code reuse. Since ASLR is not enabled on the MSCOMCTL library, this library is used for gadgets in the ROP based exploits. Hence, the ROP formulation of the exploit was able to evade both ASLR and DEP as implemented at the time. However, in either the case of traditional shellcode or ROP chains, the 2013-3906 exploits are defeated through fragmentation of ActiveX objects used to implement a scriptless heap spray. The majority of the 2013-3906 samples I observed attempt to load final malware via HTTP requests. The other exploits load the final malware in memory using the same ActiveX control loading mechanism, such that these payloads are also fragmented.

The CVE-2014-4114 vulnerability is not caused by a software coding flaw, but rather policy that allows remote code to be executed. In this vulnerability, an ActiveX control allows execution of a remote .inf file which then allows execution of a portable executable. The malware is most typically downloaded via Windows file sharing (SMB/CIFS). The vast majority of these maldocs were .ppsx files which are presentations that open automatically as slide shows. There were a small number of .docx files as well. Since this vulnerability is a policy flaw, mitigations such as ASLR and DEP do not apply. Similarly, DCR does not apply even though I fragment the OLE ActiveX controls implementing the exploit. I only block a small number of these exploits because my file fragmenter identifies them as improperly formatted.

Overall, I am able to block over 75% of the exploits in my evaluation set. If 2014-4114, which is not a traditional memory safety vulnerability, is excluded, then DCR blocks over 96% of the exploits in my evaluation set.

Table 3.2: DCR Performance

File Type	Transform Speed	Render Overhead
.doc	68.9 Mbps	0%
.docx	43.1 Mbps	2.9%

3.5 Performance Evaluation

The core performance characteristics of DCR are the time required to perform the document transformation and the overhead incurred when opening the document. The document content randomization time was evaluated by performing DCR on a number of documents. The file open overhead was measured by timing the document reader opening and rendering the document, comparing the times from the original and randomized documents. I also validated that the view of the document presented to the user remained invariant by scripting Office to open the document and print it as a PDF. I compared the resulting PDFs created from the original and modified documents to ensure equivalence in rendering. The results of the performance evaluation of DCR are summarized in Table 3.2.

3.5.1 .doc DCR Performance

To evaluate the computational expense of performing the document content randomization, I measured the time to perform this operation on a 1000 document, 249 MB, set randomly selected from the Govdocs corpus [44]. The average time to perform the document fragmentation was 28.9 seconds using a single thread on a commodity server. This equates to 68.9 Mbps of throughput in a single thread. To put this execution time in perspective, I scanned the same corpus with ClamAV which required an average 28.7 seconds to complete. Performing this content fragmentation on a single 248K sample (close to average document size) yielded an average 0.028 second execution time. The DCR operations are similar in cost to that incurred by a common antivirus engine and result in a delay that should be acceptable for most situations.

To test the performance impact of DCR on document opening and rendering, this set of

benign documents was converted to PDF using Microsoft Office and powershell scripting. There were 39 documents that were removed from this set because they required user input to open or printing was prohibited by Office. The most common cause of failing to print was invocation of protected view, which limits printing, apparently because they were created by old versions of Office (the Govdocs corpus contains some very old documents). Other obstacles to automation included prompting for a password or prompting the user as a result of automated file repair actions. In addition, following OLE file format fragmentation, an additional 125 documents opened in protected view which prevented automated printing. These files apparently triggered some file validation heuristics in Office. The same mechanisms used to break exploits can also be used for malicious intent, such as evading virus scanners. All content was present, and it was later discovered that the validation heuristic did not trigger reliably on independent formulations of the same original document–some transformations would trigger this protected view and some would not. This protection built into Office triggers on some particular block layouts but the exact criteria was not discovered by the author. If DCR is to be use widely, it would be necessary to understand and prevent triggering of this heuristic, although documents from untrusted sources (email or web) are already opened in protected view.

The test data set therefore contained 836 documents totaling 197 MB. It took about 15 minutes for the documents to be converted to PDFs which equals just over one second per document. Performing multiple trials, there was no consistent difference in speed between the original and the fragmented documents. The differences in mean open times between the original and fragmented documents was 1/50th of the 95% confidence interval. Therefore, the randomized documents take no longer to open and render. This is expected as there is no additional work required to reassemble the randomized streams. Any effects resulting from less efficient read patterns seem to be masked by file caching.

Having converted both the original and fragmented documents to PDF documents, the resulting PDFs were compared for similarity. Since the PDFs had unique attributes such as creation times, none of the PDFs generated from rendering the original documents

were identical to those generated from the fragmented documents. However, they were very similar in all respects. The average difference in size of the resulting PDFs was 40 bytes, with 513 of the PDF pairs having the exact same size. The average binary content similarity score of these derivative document pairs was 87 (out of 100) using the ssdeep utility [61]. Manual review of a small number of samples also confirmed the same content in the fragmented documents as in the original documents.

3.5.2 .docx DCR Performance

The performance impact of .docx DCR was similarly evaluated. To measure the cost of performing embedded object layout randomization, I compiled a corpus of benign .docx files from the Internet, using a web search with the sole criteria of seeking .docx files. The search yielded a wide diversity of sites with no known relevant bias on the part of the researchers.

This corpus consisted of 341 files weighing in at 76 MB. Executing my utility required an average 14.3 seconds from which I derive a single threaded bandwidth of 43.1 Mbps. Scanning the same corpus with ClamAV required 28.0 seconds, nearly double the time required for my mechanism. The time to execute on a single 225 KB document, which was an average size document in this corpus, was 0.034 seconds.

As with .doc files, I tested the impact on rendering by converting both the original and randomized documents to PDF using Office. The outcome was a mean open time of 268.5 seconds for the original documents and 276.3 seconds for the DCR documents. This 2.9% increase in document render time following document fragmentation is greater than the 95% confidence interval for these trials. This slow down is very likely due to the use of higher levels of compression in the zip container. By default, Microsoft Office uses deflate compression with the fastest compression level while the randomized compression levels are spread among four compression levels. Indeed, the corpus of randomized documents was 8% smaller than the original document set.

This performance evaluation excluded one of the 341 documents that crashed Office

post randomization. This document did not appear to be malicious in any way, but simply contained a large number of ActiveX controls that triggered a bug in Office following fragmentation. I did not determine the exact cause of this crash, but did isolate it to the fragmented OLE based ActiveX objects. Since it caused a crash instead of causing a file validation/parsing error, I do not consider it evidence of a fundamental issue with my approach, but rather a bug in Office or a special case my randomizer needs to handle.

Beyond the zip container, the vast majority of the documents in this benign corpus were not modified. Of the 341 documents, only 10 documents had OLE subobjects on which fragmentation was performed, including the crash inducing document. Since this number was so small, the user visible representation of these samples were validated manually. Both the original and the modified document were opened and compared. Barring the aforementioned single document, randomizing the OLE objects embedded in .docx files maintained the integrity of the original document as presented to the user.

For both .doc and .docx files, the CPU time required to perform document randomization is reasonable–comparable with that of signature matching based detectors. The overhead on document open is negligible. I observed an issue with heuristic detections triggering protected view in about 12% of .doc files. I also seemed to trigger a bug for a single .docx file. Barring these exceptions, the transformed documents provided the same display to the user as is produced by the original.

3.6 Content Randomization Evasion

Document content randomization is effective against many exploits created without knowledge that it would be used. If it is to remain effective following wide-scale deployment, it must be resilient to evasion.

The strength of malicious payload fragmentation lies in the number of fragments required for the payload. For fragmentation to be effective, the size of the malicious payload must be larger than the fragmentation block size.

The OLE containers used in .docx heap sprays employ a default block size of 64 bytes

which is much smaller than the shellcode required for a meaningful exploit. In most of the examples I observed, the shellcode was approximately 500 bytes in length.

Table 3.3 lists the names and sizes of shellcode components provided in the Metasploit Framework[4]. Some of these shellcode components are intended to be combined with other components to implement full shellcode functionality. For example, the block_api component provides access to the windows API via hashes as identifiers. Most of the single and stager shellcode items implement a practically useful shellcode chain. The average size of all of these components is 289 bytes. In most situations, these shellcode blocks will be extended a small amount with exploit specific register setup and shellcode encoding. The size of the larger shellcode components is on par with the approximately 500 byte shellcode observed in the .docx scriptless heap sprays. All but the smallest shellcode components would require multiple 64 bytes content blocks. The smallest components provide simple building blocks that are not useful on their own. For example, the block_exitfunk component provides functionality to terminate a process and requires the inclusion of the block_api component to function. Shellcode that provides enough malicious content to be useful in a real exploit is invariably larger than can fit within the 64 byte default size restriction imposed content fragmentation in these examples.

Current exploits are not resilient to malicious payload fragmentation because it is not currently widely deployed. However, the documented countermeasure to limits on payload size is to perform an egg hunt per payload block, which has been styled omelette shellcode [21]. Omelette shellcode locates and combines multiple smaller eggs into a larger buffer, reconstructing a malicious payload from many small pieces. The omelette approach adds at least one more stage to the exploit, in exchange for accommodating fragmentation of the malicious content.

A typical heap spray involves filling a portion of the heap with the same malicious content repeated many times, with each repetition being a valid entry point. This approach would be altered for an optimal omelette based exploit. One would spray the heap with the

[4]http://www.metasploit.com/

Table 3.3: Metasploit Windows Shellcode Sizes

Name	Size (bytes)
stage_shell	240
stage_upexec	398
single_shell_hidden_bind_tcp	341
single_service_stuff	448
single_shell_reverse_tcp	314
single_shell_bind_tcp	341
single_loadlibrary	190 + strlen(libpath)
single_exec	192 + strlen(command)
single_create_remote_process	307
createthread	167
apc	244
executex64	75
migrate	219
block_service_change_description	448
block_service	448
block_exitfunk	31
block_api	137
block_create_remote_process	307
block_service_stopped	448
stager_sysenter_hook	202
stager_reverse_tcp_rc4	405
stager_reverse_https_proxy	274
stager_bind_tcp_nx	301
stager_reverse_ipv6_tcp_nx	298
stager_reverse_https	274
stager_reverse_tcp_nx	274
stager_bind_tcp_rc4	413
stager_reverse_tcp_dns	274
stager_reverse_tcp_nx_allports	274
stager_reverse_tcp_dns_connect_only	274
stager_reverse_http	274
stager_reverse_tcp_rc4_dns	405

omelette code solely, then load a single copy of the additional shellcode eggs into memory outside the target region for the spray.

When multiple egg hunts are used to defeat malicious payload fragmentation, then the primary mitigation power is shifted to the size of a block in which the reassembly code

must reside. Each egg containing the partitioned payload could have an arbitrarily small size with a few bytes overhead for a marker used to locate the egg and an identifier to facilitate proper re-ordering. The size of the omelette code is invariably the bottleneck of the technique. If the omelette code can fit fully within a fragmentation block, then malicious payload fragmentation will not be effective.

Therefore, for omelette shellcode to operate, it must be loaded in a single 64 byte block or it will be fragmented and re-ordered. Most openly available examples of omelette shellcode, which are designed specifically to be as compact as possible, are about 80-90 bytes [27]. Of course, it may be possible to shrink the size of the omelette functionality in a given exploit, and probabilistic attacks are possible.

However, if the 64 byte block size provides insufficient fragmentation, this block size could be dropped to a level rendering any sort of egg hunt infeasible. The size of these blocks in OLE files is tunable. It is also noteworthy that the cutoff between normal and small block streams can be changed and that the block size for the normal streams is also tunable. Ergo, this flexibility in size applies generally to both normal and small OLE streams. Due to the arbitrary tuning of OLE block sizes, it is not feasible to prevent malicious payload fragmentation by shrinking the payload size using techniques such as omelette shellcode.

In exploring malicious payload size limitations, I use shellcode because methods such as omelette shellcode are relatively well documented. The same general principles apply to other situations such as ROP based exploits. Typical ROP chains are similar in size to the shellcode, so the fragmentation of DCFR is equally effective. The ROP chains I saw in the CVE-2013-3906 heap sprays were about 1000 bytes in length. Therefore small block OLE fragmentation should be able to disrupt ROP chains as well, even if omelette style techniques are employed. The same arguments should apply to .doc file level content randomization. To the degree that exploits cannot implement malicious payload reconstruction mechanisms, then file level content randomization will remain effective.

Because document content randomization is not used widely, no examples of malicious

documents could be found in the wild that used countermeasures such as omelette code. However, observations made during the manual validation performed for current exploits indicate that DCR would still be successful.

In my study of Office documents, I saw a relatively small number of exploits that were defeated by encoding based content randomization. I observed no attempts to counter this exploit protection, and there is a dearth of studies that apply to DCER evasion. As such, counterevasion strategies are necessarily speculative.

One likely DCER evasion approach would be to anticipate the encoding and adjust the payload accordingly. Some encodings are so simplistic that they could be defeated by preparing the malicious payload so that it appears as desired post encoding. For example, if base64 were a possible encoding, it would likely be possible to prepare a malicious payload that was operable following encoding despite some restrictions in content [74]. This approach would be more difficult with encoding mechanisms such as compression which have greater complexity. Even if attackers were able to circumvent the tighter constraints caused by compression, an arbitrarily large number of compression representations are possible because of the latitude afforded in compression algorithms such as deflate. Adding a custom, entropy infusing, compressor to the existing DCR mechanisms would be operationally feasible.

Assuming there are enough possible encodings to make brute forcing infeasible, the indirect approach, analogous to omelette shellcode, would be to implement a decoder. If a very small decoder can be created then it might be used to decode a larger payload. Trivial encodings such as hexadecimal or base64 may well be possible to implement in a very small decoder. Assuming an encoding method such as deflate compression is used, it is not likely that a sufficiently small decoder can be created to make this method worthwhile. I studied the compiled size of a few common decompress only deflate implementations designed specifically for small size[5][6][7]. Table 3.4 shows the size of these implementations

[5]https://code.google.com/p/miniz/
[6]https://bitbucket.org/jibsen/tinf
[7]http://www.zlib.net/

Table 3.4: Deflate Decompressor Size (Linux)

	Object Size (KB)	
Name	32 bit	64 bit
tinf tinfl100.c	5.1	6.3
zlib puff.c	5.4	6.9
miniz tinfl.c	15	18

compiled for 32 bit and 64 bit Linux. The minimum size of these implementations is about 5 KB. When compared with other decoders used in exploits, these sizes are very large. It seems that scenarios where using an over 5 KB decoder is useful for defeating content encoding based would be rare.

When attacked directly, DCFR's strength is driven by minimum fragment size which drives the number of fragments and the resulting number of possible permutations. It is not feasible to drop the size of a malicious payload small enough to evade the granularity provided by DCFR in OLE files. DCER's evasion resistance lies in both the constraints imposed by the encoding techniques employed and the number of possible encodings. It seems that the flexibility provided by encoding, especially compression, should allow sufficient entropy to make defeating DCER infeasible.

3.7 Content Injection and Memory Displacement

I explored alternatives for influencing process memory in Office using modifications to the document data itself, instead of the document container. For example, I attempted rearranging subcomponent reference or XML tag order, but could not find any practical way to influence memory load order without changing the representation of the document. A less sophisticated approach is to add non-visible content to consume memory and shift subsequent memory allocations. I sought to confirm that I could influence memory layout by injecting additional content.

Specifically, I sought to defeat .docx heap sprays by displacing them with benign content. To decrease the concentration of the heap spray, one injects benign content in the midst of

the malicious content. Because most documents have a relatively small amount of content compared to heap sprays, which are often 10s of MB or larger, large amount of benign content must be added to the heap.

This approach, while functional, has many limitations. The primary shortcoming is that it requires loading a large amount of useless content into memory, which comes at great cost. Heap sprays have very poor performance, in and of themselves. To be effective, one has to drown out the malicious content with even more inert content. This insertion of content must occur on all documents, including benign ones, for this technique to be effective. However, maldocs are free to create very large heap sprays. I sought for alternative mechanisms to dilute heap sprays, but I could devise no way to consume virtual memory space without incurring resource sapping data copies into memory. Using this technique incurs considerable overhead and is probably often not feasible due to performance impacts.

In addition to the cost of all the benign content needed to dilute a heap spray, there are other challenges. For this technique to be effective, the benign content must either displace or be interspersed with the malicious content. Proper alignment can be challenging because the part of the document that implements the heap spray cannot be known a priori. In the case of Office 2007 .docx files, adding additional content required changing some semantic meaning of the document, unlike OLE file fragmentation or OOXML zip re-encoding.

To demonstrate the effectiveness of content based memory consumption, a utility was created to modify .docx files such that when opened, the heap was sprayed with benign content. This method of spraying inert content without use of scripting is similar to that used in contemporaneous exploits [4]. This technique was pioneered in CVE-2013-3906 exploits, but has since been used in connection with other vulnerabilities and file formats [66].

ActiveX controls, with a large amount of benign superfluous content, are inserted into the document at random locations in the document. These items are set to be hidden so that they do not impact normal content. No method of adding benign content to the heap without making additions to the document could be found. However, under normal conditions, these items are hidden so the document does render as usual.

Inspection of the memory using dynamic analysis showed that the benign and malicious sprays were both loaded in memory. The individual benign and malicious objects were shuffled together such that the two sprays were interspersed according to the location of these objects in the document.

After performing manual validation via dynamic analysis on a small number of CVE-2013-3906 samples, it was confirmed that the malicious heap sprays were diluted as the ratio of exploit success matched that expected. For example, when the benign content was equal in size to the malicious heap data, the exploit success rate was consistent with the estimated rate of 50%. When the benign content was increased, the exploit success rate dropped proportionally. I found that ensuring proper alignment and therefore optimal mixing of the benign and malicious sprays is challenging, resulting in deviations from the anticipated exploit success rate.

Adding content to influence memory layout resulted in generally poor performance. Due to compression and the ability to reference data stored in the document multiple times, it was possible to incur only minor file size increases. However, to influence memory layout, memory had to be used, and the processing to consume the memory space was expensive. I found that optimal memory displacing benign objects added about 10 seconds of load time per 100MB of memory filled.

The performance characteristics of influencing memory layout through memory consumption, the difficulty of ensuring that benign content is interspersed with malicious content, and the difficulty of overpowering potentially large heap sprays with benign content make heap spray dilution impractical. Document content based mitigations using displacement would be similar to operating system based memory protections that instead of using virtual memory based techniques to implement ALSR, used actual allocations that consumed memory to introduce entropy in memory addresses.

Despite high cost, interspersing malicious content with inert content is possible using document content modifications. The ability to dilute heap sprays and drop exploit success rates was demonstrated. Since it is relatively easy to ensure that inert objects are loaded

early in the document, reliable shifts to the heap are possible. While the document had to be modified, the objects used to consume memory were marked as hidden, such that they did not change the rendering of the document.

It is possible that other exploit tactics, including memory corruption, object confusion, use after free, or other ASLR bypass techniques may provide situations where a limited content based memory dilution or shift could break exploits that would not be mitigated by OS level mitigations [25]. If situations are found where a small shift in the heap is useful for defeating exploits, the memory consumption based technique could be useful in practice.

3.8 Discussion

Not all exploits are directly impacted by DCR and some vulnerabilities may be formulated to circumvent DCR. For example, the malicious documents foiled through OLE file randomization could be modified to load the final malicious executable through a web download instead of extracting it from inside the document file. Similarly, the OOXML documents defeated through memory content location randomization could use a scripted heap spray instead of relying on document content loaded into memory. However, these changes might cause the exploit to run afoul of additional mitigations such as restrictions on ability to download executables or restrictions on the execution of macros. Hence, DCR is enabled by environmental controls such as restrictions on web downloads, Office based protections such as disabling of scripting, and operating system controls such as DEP. If these complementary protection mechanisms are not used, DCR will not be as effective. To the degree that security controls that drive attackers to use raw file content become more prevalent, DCR should increase in applicability, including in other file formats.

Some forms of DCR are more difficult to circumvent than others because they operate much earlier in the exploitation process where the attacker has lower control over the system. For example, DCR that defeats heap sprays is more resilient than that which disrupts egg hunts that extract the final malicious payload. In my evaluation, the older exploits were interrupted later in the exploitation process while the newer exploits occur much

earlier. It appears that complementary mitigations in the operating system (ALSR and DEP) constrain exploit authors to use document content earlier in the exploits.

DCR is an attractive mitigation technique because it incurs a very low performance impact. Transforming the document requires roughly the same computational resources that are already commonly employed to perform signature matching on both network servers and client programs. DCR incurs a very small performance penalty when the transformed document is opened because this mechanism leverages the file stream reassembly routines already executed by the document reader.

Just as virtual memory mechanisms enable ASLR with little overhead, the parsing and reassembly that enables multiple file level representations of the same logical document allows for efficient DCR. Any situation where data is referenced indirectly, providing for multiple possible low level representations, could potentially be used to implement exploit protections similar to DCR. I focus on content fragmentation because the file formats studied here support a large degree of layout changes. Content encoding randomization is only effective in a small number of Office exploits. However, other document and media formats might not support the same level of data fragmentation but may support arbitrary encoding or compression. The PDF format is a good candidate for file level DCER to prevent raw file reflection based malware retrieval. There is an opportunity for studying the limits of DCER, especially in document formats such as PDF where there are multiple options for encoding, the encodings can be combined for the same stream, and encoding mechanisms themselves can be tweaked. For example, instead of using standard compression levels for the deflate method, one could use probabilistic Huffman coding trees and randomized use of LZ77 data deduplication. Operating system based encoding or data randomization techniques generally have been unsuccessful due to computational overhead and the difficulty of deploying the technique which requires modifying system libraries as well as applications. However, DCER has the potential to be computationally feasible because the content encoding already occurs.

DCR is likely to be employed in situations where many multiple repeated exploitation

attempts are not easy, lowering concern of probabilistic attacks. For example, document based attacks usually require the user to take an action to view the document. Because of how client applications are used, probabilistic attacks requiring numerous attempts, similar to those employed against network daemons to defeat ASLR, are not likely to be possible.

While DCR does not impact the content of the document as interpreted by the document reader and viewed by the user, it does change the raw document file. This could potentially impact signature matching systems that operate on raw files instead of interpreting as the document reader does. Also, cryptographic signatures such as those used in signed emails would not validate correctly on the transformed document. Solutions to these issues have yet to be elaborated, but potential solutions are promising. For example, signature matching systems can implement file parsing. Signature validation systems could operate on an invariant logical representation of the parsed document, instead of a potentially arbitrary file level representation.

Chapter 4: Structural Feature Based PDF Malware Classifier

I recognize that preventing all exploitation is not possible. Even when exploits can be prevented, it is often advantageous to detect attacks so that current threats can be comprehended. Because of this, I sought to improve detection in commonly exploited file formats. Seeking to complement existing detection mechanisms, including signature matching, I studied machine learning based approaches. I found that features based on structural and metadata elements were robust for classifying PDF documents. Evaluating multiple learning algorithms, I found Random Forests to be effective. I named this PDF malware classifier PDFrate.

In evaluating my approach, I found that PDFrate provides high classification rates through cross-validation and on malware not included in the training set. PDFrate also separates commodity crimeware from malware used in targeted attacks. PDFrate is implemented as a free online service.

4.1 PDF File Format

The PDF file format is documented in ISO 32000[1]. PDF documents are based on the hierarchical organization of many objects. For example, documents are made up of pages that can contain forms, images, fonts, and text. The relationships between objects are tracked through references based on object identifiers. The PDF format utilizes various text based markers to delineate objects contained in the document and their relationships. For example, pages are contained in objects proceeded by a /Page marker, fonts exist within a /Font object, and javascript objects are denoted with a /JS or /JavaScript tag. Appendix A demonstrates the structure of two PDF documents.

[1] http://www.iso.org/iso/catalogue_detail.htm?csnumber=51502

While the structure of a PDF document is text based, the format allows binary data streams such as images to be embedded in the document. Furthermore, the PDF format allows data to be compressed and encoded with a large number of methods. PDF documents also permit inclusion of dynamic content based on javascript, forms, and other media formats such as Flash. Malware authors use the flexibility of the PDF format to advance their exploits and evade detection.

4.2 Feature Extraction

I implemented my own routines for reliably extracting metadata and structural information from PDF documents to ensure computational efficiency and resiliency in the face of malformed documents. Regular expressions are applied to the raw document to identify and extract data for further processing, if necessary. Many of the features can be derived from simple string matching. For example, the number of font objects is determined by counting the number of /Font markers. Some features require simple comparisons of string matches. For example, the size of stream objects is determined from the relative location of the start of stream and end of stream markers. Many features required extracting specific data for further normalization or processing. Examples include the dimensions of a box object or the number of lower case characters in the title. This software functions without significant PDF structure parsing or validation. This approach is beneficial when dealing with malformed documents, and it provides strong performance. This fast and loose metadata extraction intentionally results in some inaccurate or ambiguous extracted data, but the end features are deterministic. For example, a PDF edited with incremental updates may have extracted features that report an inflated object count. Additionally, instead of trying to determine the correct value in the case of multiple instances of a metadata item repeated in a document, other features such as the number of times the values differ are used. Feature extraction and classification works well even on encrypted documents because in PDF documents each object/stream is encrypted individually, leaving structure and metadata to be extracted the same as normal documents.

The majority of the metadata items are inherently numeric. The other features are all extracted or transformed to make them numeric. There are largely no differences in how binary, discrete, and continuous data is handled, although some of each type exist.

4.3 Feature Selection

I selected a large number of features to characterize PDF documents. My aim was to provide strong classification quality, including the ability to reliably distinguish targeted attacks from opportunistic attacks. In addition, the approach taken here seeks to be resilient to differences in threats and vulnerabilities by focusing on patterns in documents that apply broadly. Therefore, features are derived either from PDF document metadata or structure.

In my approach, the extracted features are designed to eliminate reliance on specific strings or byte sequences. For example, when dealing with data that might represent artifacts of specific actors, such as the author metadata item, abstracted features, such as the number of characters in the author field, are used. Similarly, features were intentionally avoided that are tightly related to specific vulnerabilities, but which have little general application, because including these features could result in strong classification for known attacks while yielding low detection rates for novel attacks.

The philosophy for feature identification was to generate as many features that parameterize the metadata and structure of the document as possible, without short-sighted regard for usefulness of individual features in discriminating between document classes. The features reflect properties of the metadata, such as the count of the characters in each metadata field; objects/streams, such as the size and count of each; boxes and images, such as the size and location of each; data encoding methods, such as use of each data encoding method; and object types, such as count of encryption objects. In total, 202 features were chosen for use.

A brief description of all the features are given in Appendix B. The names of the top features are shown in Figure 5.1. More detailed descriptions are given here for a few of the features. The count_font, count_javascript, and count_js features are determined

by the number of instances of /Font, /JavaScript, and /JS markers. The count_stream_diff variable is the difference of the instances of "stream" and "endstream" markers. The relative position in the document of the last box marker (box objects are used in layout) is reported as pos_box_max. The sum of all the pixels in all the images in the document is named image_totalpx. producer_len is the number of characters in the producer metadata object and count_obj is the number of instances of the "obj" marker. Each document has a unique document identifier that should never be modified between revisions, which is called pdfid0– pdfid0_mismatch reflects the number of unique instances of pdfid0 values in a document (which is usually always one). These features are identified by either simple string matches or more complex regular expressions applied to the raw document.

Most features are taken directly from observation of the document metadata or document structure, such as the number of font objects in the document. A few of the features are further refined by transformation of one or more elements. For example, one feature is the ratio of the number of pages to the size of the whole document.

4.4 Machine Learning Algorithm Selection

The proper selection and tuning of the machine learning mechanism is important for the overall performance of my approach. Both Support Vector Machines (SVM) and Random Forests were evaluated. Random Forests was found to be the most effective, but Support Vector Machines was competitive.

In determining the best machine learning algorithm, both Random Forests and Support Vector Machines were studied extensively, including tuning their respective parameters. These methods were chosen for primary focus because they both perform well on the type of data and features used in this study. Also, these mechanisms perform well in other similar research. The details of tuning each method is found hereafter. Other mechanisms, such as naive Bayes and artificial neural networks were explored in a limited manner but were not studied extensively due to poor initial results.

As shown in sections 4.4.1 and 4.4.2, when both are tuned optimally, Random Forests

performs about a half of an order of magnitude better than Support Vector Machines. Random Forests is used as the primary machine learning method throughout this thesis.

4.4.1 Support Vector Machines

Support Vector Machines is a kernel method based classifier. SVMs operate by finding an optimal boundary or margin that separates the classes of the data in the training set. To make this practical on multidimensional data sets, the kernel trick, where the data is mapped into highly dimensional feature space using a kernel function, is used. The classifier model consists of the parameters of the boundary that separates the classes and the parameters of the kernel that define the feature space mapping. When a new observation is classified, the data is normalized and the boundary is applied using the kernel function based feature space resulting in the determined classification.

There are a few parameters that need to be tuned to optimize SVM. The primary option is the kernel type. Four kernel types were evaluated: polynomial, linear, sigmoid, and Gaussian (radial basis function). In addition, the various parameters for SVM and those specific to the various kernel functions were optimized. The optimal value for these tunables was found through a grid search of the various parameters using cross validation on the generic features from the training set. The 10-fold cross validation was repeated ten times per parameter permutation and the results averaged.

The overview of the optimal classification performance by kernel type is given in Table 4.1.

Table 4.1: Optimal SVM Error Rates by Kernel Type

Kernel	Error Rate
Polynomial	0.53%
Linear	0.83%
Sigmoid	0.98%
Gaussian (RBF)	1.6%

The results of the grid search for each kernel type is reported in Tables 4.2, 4.3, 4.4,

and 4.5. For brevity and simplicity, only variations in cost and gamma are shown in these tables. For the kernels with additional parameters, data presented represents the optimal value for these other parameters. In the case of the polynomial kernel, the data presented is for a coefficient of 1 and a degree of 2 while the grid search included values of 1, 0, 1 and 2, 3, 4 for these parameters respectively. For the sigmoid kernel, the values -1, 0, 1 were used for the coefficient and the results with this set at 0 are presented.

Table 4.2: SVM Tuning: Error Rates for Polynomial Kernel

	cost		
gamma	1	10	100
10e-6	39%	4.2%	2.6%
10e-5	4.2%	2.6%	1.3%
10e-4	2.6%	1.3%	0.90%
10e-3	1.1%	0.76%	0.57%
10e-2	0.62%	0.53%	0.54%
10e-1	0.68%	0.26%	0.68%

Table 4.3: SVM Tuning: Error Rates for Linear Kernel

	cost		
gamma	1	10	100
10e-6	0.83%	0.89%	0.92%
10e-5	0.83%	0.89%	0.92%
10e-4	0.83%	0.89%	0.92%
10e-3	0.83%	0.89%	0.92%
10e-2	0.83%	0.89%	0.92%
10e-1	0.83%	0.89%	0.92%

4.4.2 Random Forests

The Random Forests classification method gives the result of classification based on the output of many individual classification trees, each of which votes for one of possible classes. Each decision tree is generated from a randomly selected subset of training data. Hence, Random Forests is an ensemble classifier using bagged training data. Each node in a tree is

Table 4.4: SVM Tuning: Error Rates for Sigmoid Kernel

	cost		
gamma	1	10	100
10e-6	50%	5.0%	3.3%
10e-5	5.0%	3.3%	1.5%
10e-4	3.3%	1.5%	0.98%
10e-3	1.6%	1.6%	2.2%
10e-2	4.7%	5.2%	5.1%
10e-1	13%	14%	14%

Table 4.5: SVM Tuning: Error Rates for Gaussian (RBF) Kernel

	cost		
gamma	1	10	100
10e-6	29%	4.4%	2.8%
10e-5	4.6%	2.9%	1.8%
10e-4	3.2%	1.9%	1.6%
10e-3	2.3%	1.8%	1.7%
10e-2	3.0%	2.6%	2.6%
10e-1	4.8%	4.6%	4.6%

created by selecting a random subset of features and determining the best split at each node using the training data for that node. Furthermore, each tree is based on an independent subset of features. Lastly, during classification the votes of each tree determine the result.

The Random Forests algorithm has two primary parameters that were tuned for this study. The parameter "ntree" dictates the number of trees to grow in the classifier. In addition, "mtry" controls the number of features sampled at each node in the trees. By default, a Random Forest is constructed using the square root of the number of variables for mtry and 500 for ntree. To properly tune these parameters, multiple values were tested with the classification error being compared. For the results shown in Table 4.6, the average of 10 independent trials of ten fold cross validation was used on the training data set. The values for the parameters are given as the ratio of the default values.

Table 4.6: Random Forest Tuning: Classification Error Rates

	ntree			
mtry	.5	1	1.5	2
1	0.25%	0.23%	0.25%	0.24%
1.5	0.22%	0.22%	0.22%	0.22%
2	0.21%	0.20%	0.21%	0.21%
2.5	0.19%	0.19%	0.21%	0.19%
3	0.20%	0.20%	0.19%	0.20%
3.5	0.20%	0.18%	0.19%	0.19%

It should be noted that the variance between individual cross-validation runs was relatively high when compared to the average error rate itself. In fact, the mean of the standard deviations in the trials with the same parameters was .088%. The relatively low differences between the various tuning parameters and the relatively high variance exhibited between cross-fold validation trials indicates that the actual values chosen have very little reliable impact on classification rates.

The majority of the parameter values tested perform equally well. I select double the default for ntree and triple the default for mtry, which are among the optimal levels. These parameters are used throughout this thesis. As data sets changed, this optimization was occasionally re-tested, with this selection consistently yielding among the optimal classification rates. Since the ntree is not dependent on the data set, a constant 1000 is used for ntree, while three times the square root of the number of variables is used for mtry.

Given optimal parameters for each classifier, a summary of the classification and computational performance is given in Table 4.7. While Random Forests provides better classification, the training time is higher with the chosen parameters, than that of Support Vector Machines. Classifying unknown observations is essentially identical. Drawing conclusions from the tuning sections for the respective learning methods, SVM is more dependent on proper tuning as it has a much larger variance in the outcomes as the classifier parameters are tuned.

For this study, Random Forests is used primarily due to its edge in classification performance and much lower reliance on appropriate tuning. However, Support Vector Machines can be tuned to have classification rates that are very competitive and both are computationally efficient for the task of classifying new observations. I also found Random Forests to be more resistant to evasion attacks (see Chapter 5).

Table 4.7: Classifier Performance Comparison

Classifier	Error Rate	Train Time	Classify Time
Random Forest	0.20%	260 sec	0.04 sec
Support Vector Machine	0.53%	4.9 sec	0.04 sec

4.5 Classification Labels

In our experiments, documents are classified as either benign or malicious, with malicious being further split into two categories: opportunistic and targeted. These are abbreviated "ben", "mal", "opp", and "tar", respectively. This arrangement, which is achieved using a dual-stage classifier, is represented in Figure 4.1 To be classified as malicious, documents must exploit a software vulnerability and execute malicious code. For the purpose of this study, documents that contain text instructing the reader to perform malicious actions (wire money), that contain hyperlinks to malicious content where the user must click, etc. are considered benign. Targeted attacks are separated from opportunistic ones by factors such as victim specific social engineering and correlations with other known targeted attacks.

4.6 PDFrate Evaluation

We evaluated the performance of our classifier which uses features taken from document metadata and structure. We studied the adequacy of the features selected and the resulting classification performance. Classifier performance is demonstrated through application to the training data set, use on the independent operational data set, application to multiple

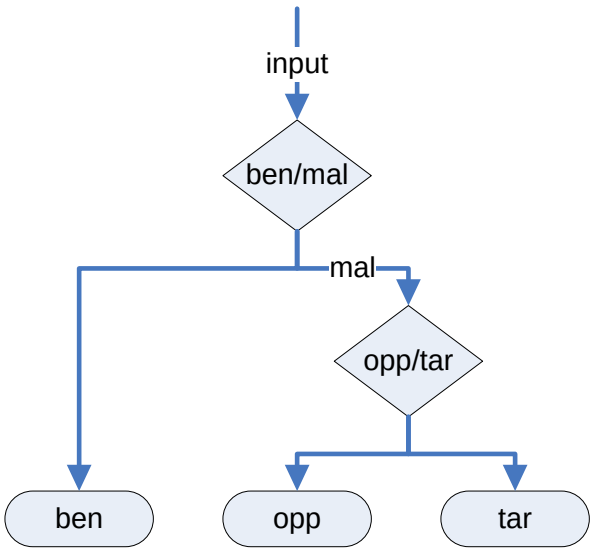


Figure 4.1: Dual Classifier Arrangement

variants, and comparison to other contemporary techniques.

4.6.1 Evaluation Data

There are two primary data sources used in this study. The first is the widely available Contagio data set [85], which is designated for signature research and testing. This data set was selected because it contains a large number of labeled benign and malicious documents, including a relatively large number from targeted attacks. This source provides a few collections of documents. All of the PDF documents from "Collection 1: Email attachments from targeted attacks" were used as targeted malicious documents. The documents from "Collection 4: Web exploit pdf (I think they all are pdf) files" are used as malicious documents. The vast majority of the documents from Collection 4 were attributed to opportunistic threats and were used as such, with a few exceptions. There were 10 identical documents in Collection 1 and Collection 4; these were considered targeted. Also, a few additional documents were identified as targeted through manual inspection and correlation to other targeted attacks. Lastly, "Collection 5: Non-Malicious PDF Collection" was used

for benign PDF examples. A total of 10,000 documents were used from the Contagio data set as training data.

The second collection is taken from monitoring a large university campus network. These documents were extracted from HTTP and SMTP traffic. The bulk of this collection was taken from approximately six days of capture. Because this data is taken from a real data feed, it is termed the "operational" data set. The operational data set required labeling by the researchers to be useful for evaluation. To separate the malicious documents from the benign, a combination of five common virus scanners were used. The corpus was scanned with the virus scanners until signature updates ceased adding detections. The virus scanner detections continued to improve until approximately 10 days following the end of the collection. Note that no single virus scanner detected all the malicious documents. All the documents that were flagged by a single scanner were subjected to additional virus scanning and manual analysis. Two of the twelve documents identified by a lone AV scanner as malicious were found to be benign. All of the malicious documents from the six-day collection were considered opportunistic, as there was no evidence to support labeling them as targeted. Eleven malicious PDFs associated with targeted attacks on the same organization, but representing multiple victims and attack groups, were added to this collection. These targeted PDFs were observed on the campus network over the span of approximately 18 months and were collected in the same manner. The addition of these targeted attacks was necessary to allow the operational data set to be used to evaluate targeted attack detection. A total of 100,000 unique documents were used from the operational data set.

In both data sets, only unique documents were used. Sampling, when it occurred, was random. Table 4.8 summarizes these data sets by displaying the number of documents of each class. Note that the training data set includes equal parts benign and malicious documents, which is desirable for training. The operational data set's ratio of benign to malicious is intended to mirror a typical operational environment and to provide insight into detection rates and false positives in the real world. The number of targeted documents is undesirably low but is the best that could be obtained given their scarcity. The operational

Table 4.8: Data Set Summary

	Training	Testing/Operational
benign (ben)	5,000	99,703
opportunistic (opp)	4,802	286
targeted (tar)	198	11
total	10,000	100,000

and training data sets are completely independent. The training set was compiled months before the operational data set.

4.6.2 Adequacy of Features

I sought to determine the degree to which the features I selected were adequate for accurate classification. In Figure 4.2, I depict the classification error as features are added to the benign/malicious classifier. The average and 95% confidence interval of the classification error of multiple randomly selected subsets of features are presented.

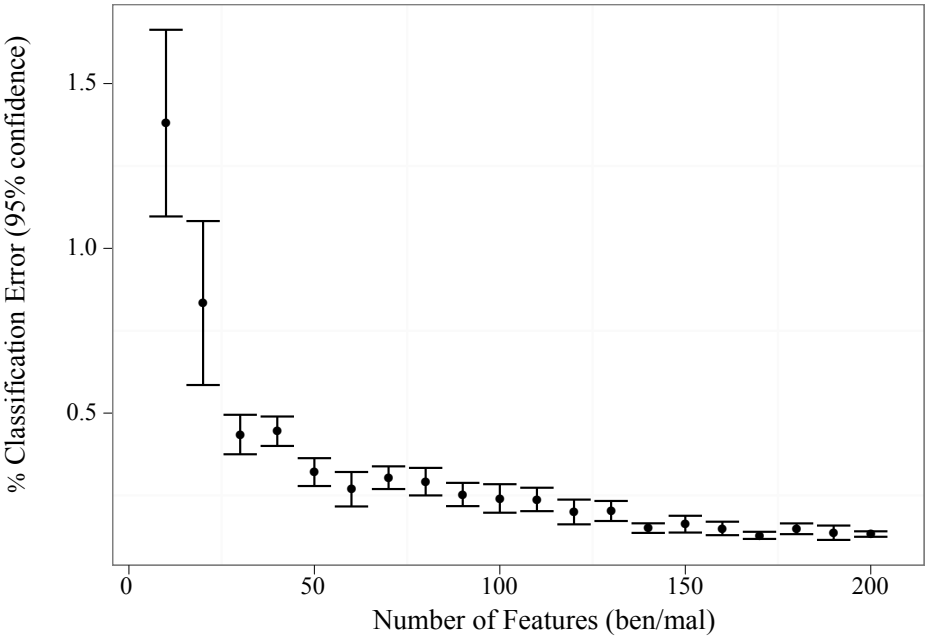
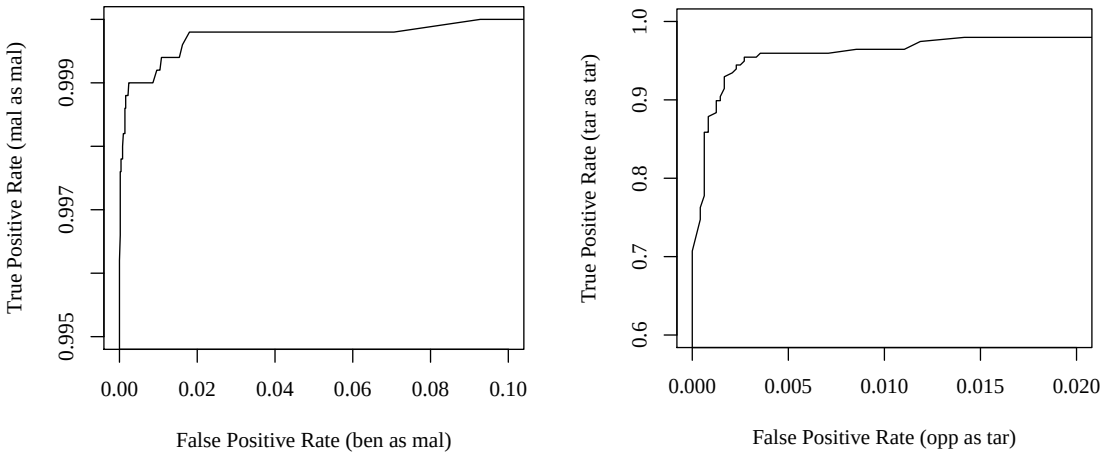


Figure 4.2: Error Decrease with Feature Count(ben/mal)

4.6.3 Classification & Detection Performance

The classifier was applied to the training data set using 10-fold cross-validation. The results from each fold are averaged to produce a single outcome for the whole set. These resulting Receiver Operating Characteristic (ROC) curves for the ben/mal and opp/tar classifiers are displayed in Figure 4.3a and Figure 4.3b respectively.



(a) ROC for Training Set(ben/mal)

(b) ROC for Training Set (opp/tar)

Figure 4.3: ROC for Training Set

In addition, Table 4.9 and Table 4.10 list select data points from these graphs. The cutoff reported in these tables is the minimum percentage of votes that an observation must exceed to be considered in the positive class (mal for ben/mal, tar for opp/tar).

Table 4.9: FP/TP Rates: Training Set (ben/mal)

Votes	FP Rate	TP Rate	FP Count	TP Count
0.7	0	0.9942	0	4971
0.6	0	0.9962	0	4981
0.5	0.0008	0.998	4	4990
0.4	0.0014	0.9986	7	4993
0.3	0.0034	0.999	17	4995
0.2	0.0076	0.999	38	4995
0.1	0.0208	0.9998	104	4999

Table 4.10: FP/TP Rates: Training Set (opp/tar)

Votes	FP Rate	TP Rate	FP Count	TP Count
0.7	0.00125	0.8889	6	176
0.6	0.00167	0.9195	8	182
0.5	0.00271	0.9495	13	188
0.4	0.00333	0.9545	16	189
0.3	0.00437	0.9595	21	190
0.2	0.00583	0.9595	28	190
0.1	0.00854	0.9645	41	191

The classifier (trained with the training set) was applied to the operational data set collected from live network observation. In lieu of presenting the ROC graphs of the classifiers applied to the operational data set, Figures 4.4 and 4.5 contain the density plots of the votes for the two classes in each of the binary classifiers.

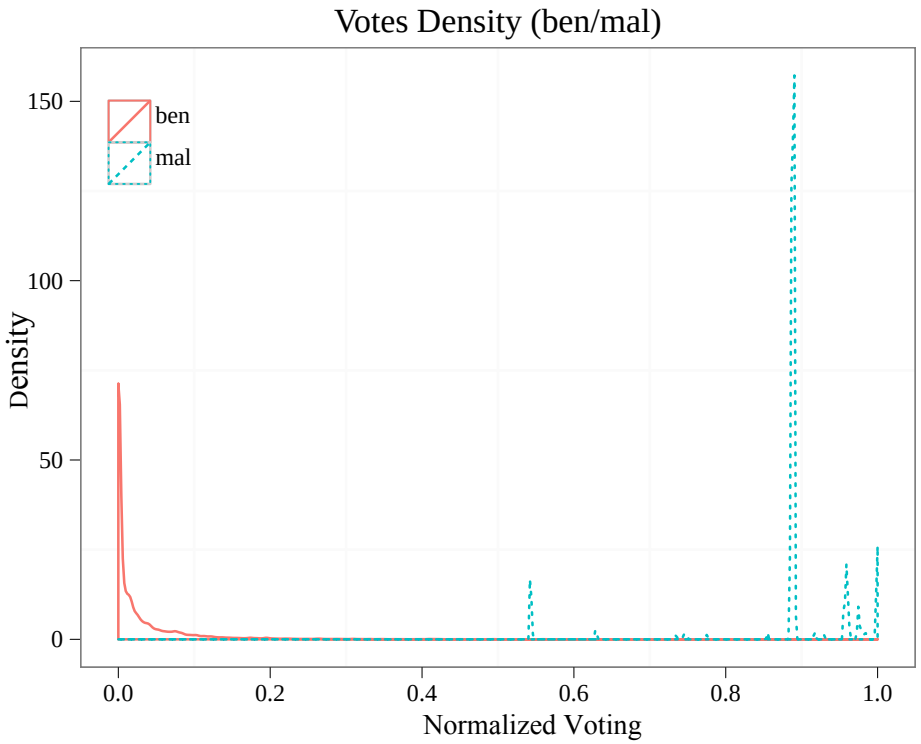


Figure 4.4: Classification Votes Density (ben/mal)

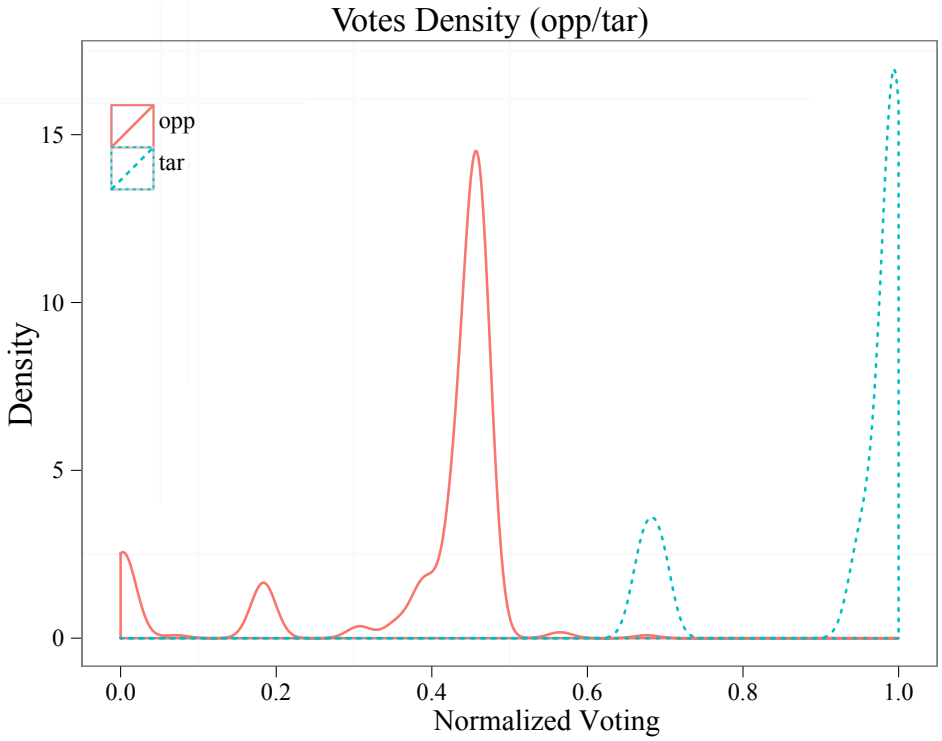


Figure 4.5: Classification Votes Density (opp/tar)

These plots show the separation between the classes in each classifier. Perfect classification would result in the negative class being a spike at zero and the positive class being located at one. Note that while the operational data set is largely independent of the training set, the classifiers still provide strong discrimination between the classes, even if some trees contribute an incorrect vote. These plots also clearly demonstrate the ability the operator has to tune the sensitivity of the classifier by adjusting the vote threshold.

In Table 4.11 and Table 4.12 I list select data points for the ROC from this data.

Table 4.11: FP/TP Rates: Operational Set (ben/mal)

Votes	FP Rate	TP Rate	FP Count	TP Count
0.8	0.00021	0.9327	21	277
0.7	0.00057	0.9461	57	281
0.6	0.00132	0.9529	132	283
0.5	0.00244	1.0000	243	297

Table 4.12: FP/TP Rates: Operational Set (opp/tar)

Votes	FP Rate	TP Rate	FP Count	TP Count
0.8	0.0000	0.82	0	9
0.7	0.0000	0.82	0	9
0.6	0.0035	1.00	1	11
0.5	0.0105	1.00	3	11

Table 4.13: Variants in Data Sets

	Training	Operational	Overlap
opp samples	4,802	286	-
opp variants	812	31	6
tar samples	198	11	-
tar variants	186	9	1

4.6.4 New Variant Detection

To demonstrate the resiliency of structural and metadata features across differing data sets, the results of the off-the-shelf antivirus scanners were used to distinguish "variants" of very similar malicious documents. The results of the five antivirus scanners were used to create a variant identifier which is the 5-tuple of the signature name of each scanner. The results of categorization of the samples into variants are shown in Table 4.13.

This categorization of samples into groups of variants resulted in a significant reduction in the opportunistic samples but a relatively minor reduction in the targeted samples. This is consistent with the wider and more automated distribution of opportunistic samples as well as the more exclusive distribution and higher likelihood to include manual modifications to support AV evasion of targeted samples.

To the degree possible to discern from the names, the AV signatures are related to the exploit and javascript used in exploitation. There were relatively few signatures that appeared to be related to the actual malware families concealed in the document. Indeed, many of the documents merely contain shellcode which in turn downloads specific malware.

Of the two groups of targeted variants in the operational data set, one pair was attributed to the same persistent actor/embedded malware family, while the other pair of documents are from separate actors/embedded malware families. In both cases, the documents had very similar structure and metadata, leading to the conclusion that they were derived from the same document template.

There is a relatively small amount of overlap in the variants from the training and operational data sets. The ability of the classifier to effectively classify new variants that were not included in the training set demonstrates that the features used for classification are durable. Variations in malicious documents that require unique signatures can be detected with a common classifier based on metadata and structure.

4.6.5 Comparison to PJScan

To demonstrate the effectiveness of the mechanisms presented here and to add further validation to the quality of the data sets used, the results of using PJScan are presented here. PJScan is not designed to separate targeted from opportunistic attacks. Thus, I only used benign/malicious classification to separate malicious from benign documents regardless of the type of threat. The classifier was trained on the 5,000 malicious documents in the training set with default parameters. Results are shown in Table 4.14. PJScan only returns a result for documents from which it can extract javascript. Therefore, in our experiments, I only count the number of documents for which a result is returned. The classification error rate is given for those documents for which a result is returned.

Table 4.14: PJScan Results

Data Set	Class	Classified	Not Classified	Classification Error	Detection Rate
Training	ben	5%	95%	1%	-
Training	mal	85%	15%	9%	78%
Operational	ben	3%	97%	1%	-
Operational	mal	17%	83%	36%	3%

PJScan was unable to classify many malicious documents. Manual analysis reveals

that many of the malicious documents PJScan cannot analyze have javascript in them but javascript is in atypical locations. Javascript code inside document metadata sections or inside corrupted document structures is not correctly parsed by PJScan, which is a known limitation. It appears that the newer operational data set has a much higher prevalence of these conditions, which prevent successful analysis. PJScan provides decent results for the training set, but when the trained classifier is applied to the operational data set, the quality of classification drops dramatically.

The biggest limitation of PJScan applied to the data sets in this study is the inability to successfully extract the features used for classification. The mechanisms presented here, which use simple signature matching without document parsing or decoding, compare favorably because they can be more reliably extracted in practice. The mechanisms presented here also compare favorably when considering the adequacy and durability of the classifier when applied to the training data and extrapolated to other, independent data sets.

4.6.6 Computational Complexity

The document classification process can be divided into three logical steps: feature extraction, classifier training, and classification of new observations. Feature extraction must occur for both training data and new data to be classified. The majority of the processing in feature extraction is dedicated to matching signatures on the documents. This facet was poorly optimized in the implementation used for this study where multiple signatures were applied to the document serially, with each of the signatures requiring another pass through the document. This implementation could be improved by making this signature matching parallel, which would improve performance approximately an order of magnitude and put performance roughly on par with conventional antivirus scanners.

Once the features are extracted from documents to be classified, running these observations through the classifier is extremely fast. Training the classifier is more expensive, but this only needs to occur infrequently. Table 4.15 demonstrates the run times of these

operations applied to the training data set, which contains 10,000 documents. The experiments were performed on an Intel Xeon X5550 processor running 2.67GHz CPU. All the applications were executed in single-thread mode.

Table 4.15: Run Times on Training Data

Operation	Time
Feature Extraction	14 min
Classifier Training	38 sec
Observation Classification	1 sec

Similarly, little effort was placed into minimizing use of memory. However, for all operations, memory usage was negligible except for training the classifier, which required about 1 GB of RAM.

4.7 PDFrate Online Service

To better study adversarial learning, I provided a publicly accessible online implementation of PDFrate[2]. The goals were to collect additional PDF based malware samples, including novel malware and evasion attacks, and to observe attempts at other forms of adversarial learning, including training set poisoning.

The PDFrate website provides scores from classifiers based on multiple training sets. The Contagio data set is taken from a widely available data set designated for researchers [85]. It contains 10,000 documents, evenly split between benign and malicious. The list of documents in this set is provided such that this training set can be replicated. I compiled the University data set from various sources. It contains over 100,000 documents. There is another classifier, based on a training set created by the PDFrate user community. The PDFrate service allows users to provide labels for individual documents and these labels are used in construction of this Community classifier.

During operation from September 2012 to March 2016, over 50,000 unique documents

[2]http://pdfrate.com

were submitted to PDFrate. The vast majority of these documents are associated with external studies [23, 70, 71, 107, 120]. PDFrate has been among the most popular malware detectors used in recent adversarial learning studies. While the PDFrate service was very effective at attracting practical mimicry attacks, there were few attempts at adversarial learning in the form of training set poisoning attacks. The PDFrate online service received under 100 community labels. This small amount of data makes any studies of attacks against the community driven training set impractical as that training set is not sufficiently differentiated from other training sets.

Chapter 5: Countering Adversarial Learning

I studied the operation of PDFrate in an adversarial environment extensively. I evaluated synthetic attacks targeting the top features and demonstrated how to counter these attacks by introducing noise in the training set. Since these attacks operate on feature vectors instead of real documents, it is not clear if they are practical to execute.

As a result of making PDFrate available for public use at pdfrate.com, there were multiple studies that sought to defeat PDFrate [23, 70, 71, 107, 120]. These studies implement state of the art attacks against PDFrate. These attacks include mimicry attacks (addition of benign attributes), reverse mimicry attacks (mimization of malicious attributes), and feature extractor subversion.

I study ensemble classifier introspection. Mutual agreement analysis produces a per observation confidence estimate by measuring the amount of coherence in voting. I find that ensemble classifier mutual agreement analysis detects most misclassifications, including evasion attempts and previously unseen malware. My approach is also effective in optimizing retraining of the classifier. We find that feature bagging is important to building a classifier with enough diversity to withstand evasion.

5.1 Top Feature Mimicry Attack

It is important that any detection mechanism demonstrate resistance to intentional evasion. Therefore, the robustness of the selected features under mimicry and evasion attacks is crucial to the actual detection rates that can be achieved in a real-world environment. The detection mechanism presented in this thesis is designed to classify documents based on similarity to past documents of the same class. These similarities between documents can arise from a wide spectrum of root causes varying among necessity, convenience, convention,

and ambivalence. Presumably, some of the attributes of malicious docs are easy to modify and others are more difficult. For example, while use of javascript is often not strictly required for exploiting vulnerabilities in PDF readers, it is often the most practical method for triggering many exploits. Hence the features related to the existence of javascript may be hard for attackers to modify. Alternatively, it may be trivial to spoof or remove metadata such as the producer field. It is infeasible to fully enumerate or to accurately predict or anticipate all the methods used to attempt evasion, especially as some of the factors are dependent on the attacker and attack vector. Some constraints on evasion are also caused by the use of this mechanism in parallel to other techniques.

Note that the mere existence of benign elements or lack of specific malicious artifacts is not sufficient to evade detection. Some of the malicious documents evaluated in this study contained large portions of benign content. This indicates that the malware packager either intentionally added benign elements to a malicious document or that the malware was added to an existing benign document. Conversely, many malicious documents are devoid of optional metadata. Reliable classification, regardless of the existence of coincidental features, is critical.

5.1.1 Mimicry Attack Effectiveness

One likely evasion technique I anticipate is a mimicry attack where malicious documents are purposefully modified to normalize some of their features and make them similar to benign documents while still retaining the embedded malicious content. If the attacker has knowledge of specific features used in the classifier and their importance, along with a good representation of what the defender considers as normal, the attacker can focus on mimicking the features most important for classification.

To simulate mimicry of document properties, I modified the top ranked features of malicious observations and subjected these modified observations to the classifier. For simplicity's sake, the documents themselves are not actually modified, but rather the previously extracted feature sets are modified. Specifically, the mean and standard deviation of the

benign observations is calculated and the values for the malicious documents are replaced with random values that fit a normal distribution with the same mean and standard deviation. Note that this method may result in doctored features that are inconsistent or illogical. The six most important features, as ranked by the mean decrease in accuracy measurement, were selected for evasion testing. These features are ranked above the others with some amount of separation, as shown in Figure 5.1.

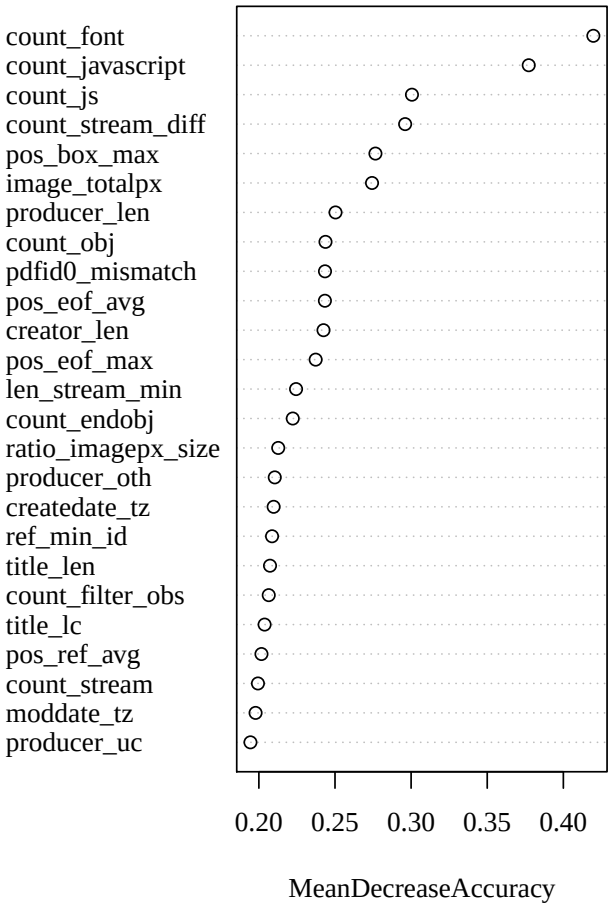


Figure 5.1: Feature Importance

By causing the malicious samples to mirror the top six features of the benign, the classifier error rate can be raised a great degree, as shown in Table 5.1. The average of the results of five independent trials using 10-fold cross validation is presented.

Table 5.1: Mimicry: Classifier Error Increase

Features Mimicked	Classification Error (%)
None	0.14
count_font	12.26
(+) count_javascript	17.04
(+) count_stream_diff	20.01
(+) count_js	20.07
(+) pos_box_max	22.18
(+) image_totalpx	22.30

By manipulating the most heavily used or distinctive features, it is possible to severely curtail the detection capabilities of the classifier.

5.1.2 Introducing Noise to Counter Top Feature Mimicry

The best reaction to changes in document attributes leading to misclassification is to retrain the classifier, causing the classifier to adjust how it treats the mimicked features. If retraining the classifier is not adequate to raise classification rates to an acceptable level, additional features can be discovered and used instead. This tactic is reactionary at best and cannot ensure detection of documents that are very dissimilar to historical examples of documents of the same class. To be able to detect intentional evasion, proactive measures must be taken.

An obvious reaction to mimicry attacks on the features heavily employed by the classifier is to remove them altogether and rely on the other features. An important distinction is that variable importance, as reported by Random Forests, is an indication of the value of the feature as used in the classifier. However, that a feature has a high importance does not necessarily mean that the feature is useful for classification on its own, nor does it mean that the classifier has to rely heavily on that feature for successful classification. Table 5.2 shows the increase in classification error as the top features are removed.

Removing the top ranked features has a surprisingly low effect on classification error because so many other useful features are retained. If the attacker is able to only modify a

Table 5.2: Classifier Error with Features Removed

Features Removed	Classification Error (%)
None	0.14
count_font	0.21
(+) count_javascript	0.28
(+) count_stream_diff	0.28
(+) count_js	0.29
(+) pos_box_max	0.29
(+) image_totalpx	0.29

few attributes of malicious documents, and the defender is able to anticipate these, removing features may be an acceptable countermeasure. It is desirable to be able to counter evasion without fully negating the predictive value of variables targeted for evasion. One method of achieving this result is to perturb the training set such that the resulting classifier is no longer as susceptible to evasion. The perturbation is performed by artificially modifying the features of a subset of the malicious observations in the training set to increase the variance of these features, making them less uniform. The loss of a focal point due to the increased variance reduces the importance of these features without fully eliminating them.

Table 5.3: Classification Error with Training Data Perturbation

% Perturbation	Original Data	Mimicry Data
0	0.14	26.12
0.05	0.14	14.11
0.1	0.15	9.19
0.5	0.15	1.80
1	0.16	1.13
5	0.21	0.69
10	0.22	0.52
50	0.26	0.16
100	2.06	0.12

To test the effectiveness of perturbation, the same method used to simulate evasion is used to modify a subset of the observations in the training set. The top six features of

a subset of the malicious observations is set to values taken from a randomly generated normal distribution mirroring the mean and standard deviation of the benign observations. Table 5.3 shows the results of testing using the perturbation method. The average of the results of 5 independent trials using 10-fold cross validation is presented. The training data is perturbated and the resulting classifier is used both on the remaining unmodified training data and the same training data modified to simulate mimicry evasion. The percentage of the training data perturbated is varied, demonstrating a trade-off between accuracy with historical data and evasion resistance. Hence, it is possible to defeat the top feature mimicry attacks with only a minor drop classification accuracy by perturbing the training set. This perturbation decreases the learned model's reliance on these top features.

5.2 Independent Evasion Attacks

PDFrate was the target of many published evasion studies: Mimicus [107], EvadeML [120], Reverse Mimicry [70, 71], and Parser Confusion [23].

5.2.1 Mimicus

Mimicus[1] is a framework for performing mimicry attacks against PDFrate. It is the implementation of what is described by Šrndić and Laskov as "the first empirical security evaluation of a deployed learning-based system" [107]. It is an independent, comprehensive, and openly available framework for attacks against the online implementation of PDFrate.

Mimicus implements mimicry attacks by modifying existing malicious documents to appear more like benign documents. Mimicus adds markers for additional structural and metadata items to documents. These additions do not involve adding actual content that is interpreted by a standards conforming PDF reader, but rather these additions exploit a weakness in the feature extractor of PDFrate. The extraneous PDF attributes are added in slack, or unused space, immediately preceding the document trailer (structure at the end of the document), which is not prohibited by the PDF specification. This approach

[1] http://github.com/srndic/mimicus

provides considerable flexibility in the evasion attack as the additional elements do not have to be valid. Mimicus enables a simple process for the attacker. The attacker constructs a malicious document without concern for PDFrate evasion. Mimicus then adds the necessary decoy structural elements. This mimicry attack only adds fake elements to the document file–no existing elements are removed or modified.

Mimicus constructs these decoy elements by comparing a malicious document to multiple different benign documents. The feature vectors for the malicious documents are adjusted to mirror the feature vectors for the benign documents. These adjustments are bounded by the modification approach Mimicus uses. The candidate mimicry feature vectors are run through a local PDFrate replica to determine the scores. The best feature vector is selected. That feature vector is used as the goal in modifying the original malicious document by adding decoy structural and metadata elements. Due to interrelated features and other complications, it is not feasible to construct a final mimicry malicious document that exactly matches the target mimicry feature vector. The resulting mimicry malicious document has a feature vector that is somewhere between that of the original Trojan document and that of a benign document. After the mimicry document is created, it is submitted to pdfrate.com for evaluation.

An important observation of the Mimicus study is that the interdependency of PDFrate's features make mimicry attacks more difficult because modifying one feature necessarily affects other features. It is generally accepted that irrelevant or redundant features are not desirable for machine learning methods. However, in the case of PDFrate, redundant features appear to make evasion attacks, like those implemented by Mimicus, more difficult by making construction of a PDF matching a target feature vector more difficult.

The Mimicus attack model requires knowledge of the feature set used by PDFrate. The premise is that for a mimicry attack to be successful, at least knowledge of the type of features is necessary. Also, since this attack leverages a difference between normal PDF readers and the PDFrate feature extractor, knowledge of how to exploit this difference is also necessary. Hence, all Mimicus attack scenarios are labeled with an "F", indicating that

the attacker used knowledge of the feature set.

Relying on the common basis of the feature extraction, the Mimicus attacks demonstrate various levels of knowledge used by the attacker. In situations where the training data and classifier are known by the attacker, replicas that are very close to the original are used. When an attacker with limited system knowledge is modeled, reasonable substitutes are employed. The labels "T" and "C" are used to denote attacker knowledge of training data and classifier, respectively. Hence, an attack scenario with the label "FTC" denotes attacker knowledge of all three major facets of PDFrate.

The training set used by the Contagio classifier of PDFrate is publicly documented and is readily available to researchers. Hence, in attack scenarios where the training data is known by the attacker, the same data set is used by PDFrate and Mimicus. For scenarios where the attacker has no knowledge of the training set, Šrndić and Laskov compiled a surrogate training set with malicious documents sourced from VirusTotal and benign documents sourced from the Internet. In addition, they selected 100 malicious documents from within the Contagio training set for the baseline attack documents. To allow reproduction of results, all of the data sets used by Šrndić and Laskov are documented.

Lastly, to complete the offline PDFrate replica, Šrndić and Laskov used a Random Forests classifier when knowledge of the classifier was known, and a Support Vector Machine classifier to simulate the case of the naive attacker. The Mimicus study shows that when all three particulars of PDFrate are spoofed, the result is nearly identical scores from the PDFrate online and the Mimicus offline classifier, despite various implementation differences. Mimicus also implements a GD-KDE attack which seeks to attack the SVM surrogate classifier directly. This attack does not apply to Random Forests classifiers, and therefore does not directly apply to PDFrate.

5.2.2 EvadeML

Xu et al. studied yet another approach to evading machine learning based PDF classifiers [120]. Their approach, EvadeML[2], is inspired by evolution in living beings. Xu et al. perform randomized mutations to documents operating on PDF objects. Each mutation either adds an object from a benign PDF, removes an object from the malicious PDF, or performs an object replacement. To evaluate the mutuation, it is applied to multiple documents. A mutated document is subject to dynamic analysis to ensure the malicious behavior is retained. The evasiveness of the mutated document is evaluated by testing against a local replica of PDFrate. Multiple rounds of mutation are performed resulting in evolution traces up to hundreds of mutations in length.

Xu et al. performed random mutations on 500 seed documents. They found that their technique was able to drop PDFrate scores below 50% using traces ranging from very short up to length of 354, depending upon the document. In analyzing the modifications found to be effective in dropping PDFrate scores, Xu et al. observed that the genetic programming approach was effective in identifying features ranked highly by the classifer, but which are not high fidelity indicators of either benign or malicious documents. The stochastic mutation approach takes advantage of the same weakness that my top feature mimicry attack exploits (see Section 5.1).

5.2.3 Reverse Mimicry

Maiorca et al. also study evasion against PDFrate and other PDF document classifiers [70, 71]. They advance the Reverse Mimicry technique. Instead of adding content to a malicious document to make it appear benign (as Mimicus does), they embed malicious content into a benign PDF, taking care to modify as little as possible. The Reverse Mimicry attack implements an independent evasion approach against PDFrate.

Three different evasion scenarios are advanced by Maiorca et al. In the EXEembed scenario, a malicious executable is implanted in an existing benign PDF document. The

[2]http://evademl.org/

malware is executed when the document is opened. These documents utilize CVE-2010-1240. In the PDFembed scenario, a malicious PDF is embedded into a benign PDF. These embedded documents are rendered automatically when the document is opened. For evaluation, Maiorca et al. embedded a document leveraging CVE-2009-0927 into existing benign PDF documents. Lastly, in the JSinject scenario, malicious javascript, the same used in the PDFembed embedded document, is injected directly into the root benign document.

In order to evade detection, the Reverse Mimicry attacks focus on changing the document structure as little as possible. For example, in the EXEembed attack, a new logical version of the PDF is constructed with few new structural elements, but all the content from the original PDF is left in the file. A compliant reader will not display the content associated with the previous version of the document, but the artifacts will be analyzed by the feature extractor of PDFrate and similar detectors.

In addition to minimizing the structural artifacts of the malcode injection, Maiorca et al. make use of PDF encoding, especially stream compression, to hide the inserted content. For example, in the PDFembed attack, the malicious document is embedded in a compressed PDF stream. Detection tools, such as PDFrate, that do not decompress the PDF streams are not able to extract features from the embedded malicious PDF.

5.2.4 Parser Confusion

Carmony et al. study the ability to defeat PDF detectors through parser confusion attacks [23]. These attacks operate by exploiting flaws or limitations in PDF parser implementations. Carmony et al. focus on various popular PDF parsing libraries and tools. The confusion attacks leverage ambiguities in the PDF specification, mis-interpretations of the specification, incomplete implementations of the specification used by malware parsers. These attacks also leverage promiscuous parsing in the Adobe PDF reader, which attempts to repair and render broken or invalid documents. In addition to parser confusion attacks, Carmony et al. also implement a reverse mimicry. This approach is similar to, but more advanced than, the PDFembed scenario proposed by Maiorca et al. This technique involves

embedding malicious content in many layers of encoding including encryption techniques that are often not implemented.

Carmony et al. evaluated their approach using a single base malicious file with various parser confusion methods against multiple parsers and detection mechanisms. Since PDFrate does not analyze PDF content (only structure and metadata), PDFrate is evaluated against the combination of all parser confusion methods and their reverse mimicry implementation.

The Mimicus, EvadeML, Reverse Mimicry, and Parser Confusion attacks all use unique paths to evade PDFrate. Mimicus uses addition of decoy objects that would not be processed by a normal PDF reader but are parsed by the simple regular expression based processing of PDFrate. EvadeML uses genetic programming to find mutation traces that result in evasive documents. The Reverse Mimicry attacks, on the other hand, use valid PDF constructs to minimize and hide malicious indicators. The Parser Evasion attacks attempt to break the parsing routines necessary for feature extraction.

5.3 Mutual Agreement Analysis

An ensemble classifier is constructed from many base classifiers. To provide meaningful diversity in the ensemble, each individual classifier is constructed using mechanisms such as random sampling (bagging) of training data and features. Typically, the result is combined by voting, where each independent classifier gets an equal vote. The count of votes are summed to generate a score. If the score is over 50%, then the observation is labeled malicious. Otherwise, the result is benign.

Ensembles have been shown to improve accuracy in many use cases, including malware detection. However, we have found the primary advantage of ensemble classifiers to be that they can provide a measure of internal coherence that serves as an estimate of the classifier's confidence of individual predictions.

In a well preforming ensemble, the majority of individual classifiers provide the same vote. If the base classifiers provide conflicting votes, then the ensemble is in a state of

Table 5.4: Relative Performance of Individual Trees in Contagio Classifier Indicated as Above (+), Below (-), or Within (0) 0.5 Standard Deviations of Forest Average

Evasion Scenario	Individual Tree Performance															
F_mimicry	0	+	+	-	0	0	-	+	0	+	-	0	+	-	+	0
FC_mimicry	+	+	+	-	+	0	-	+	0	+	-	-	+	0	0	0
FT_mimicry	0	+	+	-	-	0	0	+	0	0	-	0	0	0	+	-
FTC_mimicry	-	+	+	-	0	+	0	-	-	+	0	-	+	0	+	+
F_gdkde	-	+	+	+	+	+	-	-	+	+	0	0	+	-	+	-
FT_gdkde	+	+	+	+	0	+	-	-	+	+	+	-	+	+	-	-
JSinject	+	-	-	0	+	+	-	0	+	+	+	0	0	+	0	0
PDFembed	0	-	-	+	0	0	0	-	-	-	-	+	+	-	-	-
EXEembed	-	0	0	-	-	-	+	0	+	0	-	-	-	+	0	+

disagreement and the prediction is less trustworthy. The agreement or disagreement in voting of individual contributors in the ensemble provides an estimate of the confidence of the prediction of the ensemble.

A classifier may not be able to provide an accurate response for some observations. For example, when a 50/50 vote split occurs in traditional ensembles, a prediction is provided using a method such as random selection. Most applications will treat a randomly selected prediction when the classifier is in total disagreement the same as one in which all contributors vote for the same class. However, in the case of complete disagreement, the only reasonable interpretation is that the classifier cannot make a competent prediction.

Diversity in ensemble classifiers is the core attribute that facilitates mutual agreement based confidence estimates. This diversity is caused by extrapolation in individual classifiers. Barring limitations of the classifier scheme and quality of features, when an observation is close to samples in the training set, the classification is well supported and should be accurate. However, as new observations diverge farther from training samples, the classifier is forced to extrapolate. For ensemble classifiers that employ bagging effectively, the farther new observations are from classifier training, the more disagreement there will be in the ensemble.

This diversity in extrapolation is observed in the Random Forest based classifiers used in

Table 5.5: Ensemble Classifier Outcomes

Voting Score	Outcome		Evasion Type
[0,25]	Benign		Strong Evasion
(25,50)	Uncertain	(Benign)	Weak Evasion
[50,75)		(Malicious)	
[75, 100]	Malicious		No Evasion

PDFrate. Table 5.4 shows the classification performance of the first 16 trees (out of 1000) in the Contagio classifier applied to various mimicry attacks. Performance is reported relative to the forest average number of votes for the correct class, dividing at $\pm$ 0.5 standard deviations. It is observed that the vast majority of the trees have all three outcomes depending upon the evasion scenario: average (0), below average (-), and above average (+). Hence, when applied to data distant from the training data, the accuracy of each tree varies widely among observations. There are no universally strong or weak trees. The random noise present when extrapolating far from the training data is what enables mutual agreement analysis.

To apply mutual agreement analysis generally, I propose a new outcome, uncertain, in addition to the predictions of benign and malicious. Instead of splitting the vote region in half, I split it into four quadrants. In the 0% to 25% region, the majority of the votes agree that the result is negative (benign). Similarly, in the 75% to 100% region, the majority of the votes agree that the result is positive (malicious). However, if the score is between 25% and 75%, the individual classifiers disagree and the outcome is uncertain. To support comparison with simple ensemble voting predictions, this area can be split into the other two quadrants: uncertain (benign) from 25% - 50% and uncertain (malicious) from 50% - 75%. These classification outcomes are demonstrated in Table 5.5. The uncertain rate (UR) is the portion of observations that fall within the uncertain range.

To be more precise about this concept, I introduce a metric to quantify the agreement between individual votes in an ensemble classifier:

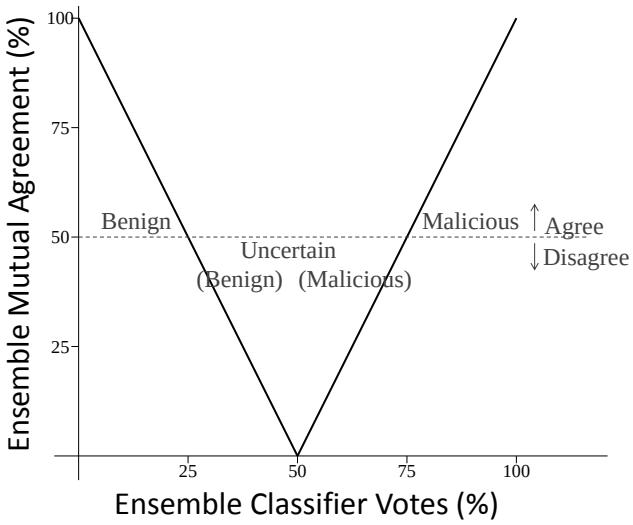


Figure 5.2: Mutual Agreement Based on Ensemble Vote Result

$$A = |v - 0.5| * 2$$

Where A is the ensemble classifier mutual agreement rate and v is the portion of votes for either of the classes. This function is demonstrated in Figure 5.2, which also shows the classifier outcomes resulting from a 50% mutual agreement threshold. The end and middle points drive the general shape of this function. If the classifier vote ratio is either 0 or 1, then the classifier has full agreement on the result and the mutual agreement should be 1 (or 100%). If the classifier is split with 0.5 of the votes for each class, then the mutual agreement should be at the minimum of 0 (or 0%). As long as a single threshold is used, it is not important what shape is used for the lines between these end and middle points– any continuous curve would allow the selection of a given threshold on the classifier vote scores. The function need not follow the distribution of scores, for example. I choose a linear function because it is straightforward.

The threshold for mutual agreement is the boundary above which the classifier is said to be in a state of ensemble agreement, and the resulting classification should be considered

valid. Below this mutual agreement rating, the classification is specious. I use the boundary of 50% throughout most of this thesis. However, this value should be adjusted by the operator. Decreasing this threshold decreases the number of observations in the disagreement or uncertain classification zone. Tuning of this threshold is discussed in detail in Section 5.7.

Mutual agreement analysis is effective at identifying the specific samples on which the classifier performs poorly. In the context of evasion attacks, ensemble mutual agreement serves as criteria for separating novel attacks and weak mimicry attacks from effective mimicry attacks. For novel attacks, it is common for the voting result to be distributed around 50%, indicating that the observations under consideration map consistently close to neither the benign or malicious samples in the training set. Since these attacks fall in the relatively rare uncertain range, they are easily discerned and are considered weak evasions. Strong mimicry attacks are those where the distribution of the attack votes is close to that of the benign observations. Hence, typical novel attacks are identified by mutual agreement analysis, but strong mimicry attacks cannot be. Since uncertain observations are supported poorly by the training set, these observations are the most effective to add to the training set in order to improve classifier accuracy.

In operation, mutual agreement analysis is employed to prevent evasion of an intrusion detection system. The mutual agreement rate is trivially derived from the result provided by an ensemble classifier at the time that detection occurs. Ensemble classifier agreement can be used in many ways by the operator, including adjusting the vote threshold to prevent false positives or false negatives, filtering observations for quarantine or more expensive analysis, and prioritizing alerts. The strength of mutual agreement analysis is that it can be used to identify probable intrusion detection evasion at the time of evasion attempts.

5.4 Evaluation on Operational Data

I applied mutual agreement analysis to PDFrate scores for documents taken from a network monitor processing files transferred through web and email. This data set includes 110,000 PDF documents, which I randomly partitioned into two data sets. The operational

Table 5.6: PDFrate Outcomes for Benign Documents from Operational Evaluation Set

	Benign		Malicious	
Classifier		Uncertain		
Contagio	98076	1408	203	40
University	99217	360	95	55

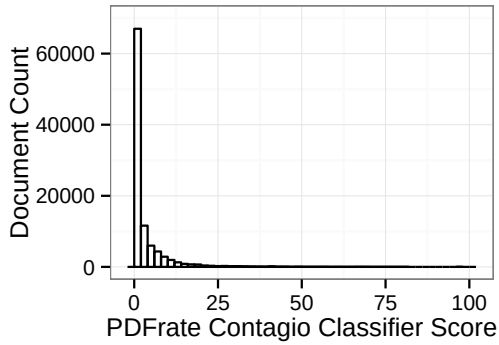
Table 5.7: PDFrate Outcomes For Malicious Documents From Operational Evaluation Set

	Benign		Malicious	
Classifier		Uncertain		
Contagio	0	0	19	254
University	0	0	0	273

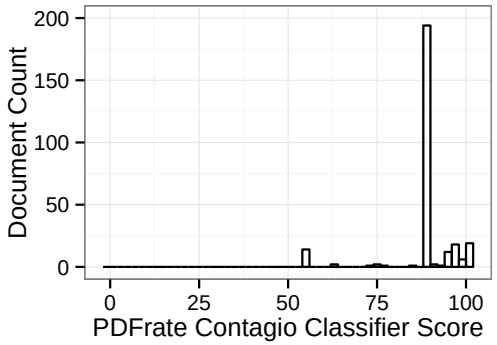
evaluation set contains 100,000 documents and operational training set contains 10,000 documents. Ground truth for the documents was determined by scanning with many antivirus engines months after collection. These data sets included 273 and 24 malicious documents respectively. Table 5.6 and Table 5.7 show the scores for the operational evaluation data set using both the Contagio and the University classifiers of PDFrate. The distribution of the PDFrate scores for the benign and malicious samples of this operational evaluation data set Figure 5.3.

It is important to note that the scores for the benign and malicious examples are weighted heavily to the far end of their respective score range, with the distribution falling off quickly. In a typical system deployment, the number of observations in the uncertain range is very small and the majority of misclassifications fall within the uncertain region. Hence, mutual agreement analysis can be used to make an estimate of the upper bound on the number of misclassifications, at least in the absence of strong evasion attacks.

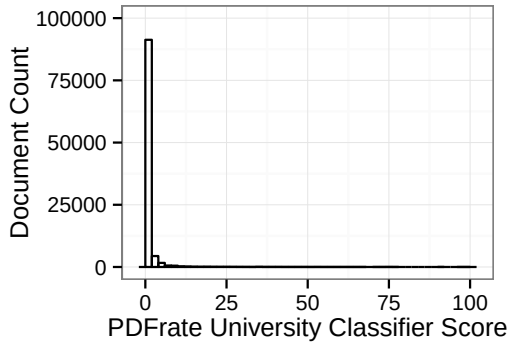
Not only is ensemble classifier mutual agreement analysis useful for identifying when the classifier is performing poorly, it is also effective for identifying specific examples that will provide the most needed support to improve the classifier. To demonstrate this, I sought to replicate improvements to the classification scores that would occur in the operational evaluation data set as additional samples are added to the classifier training set. I started



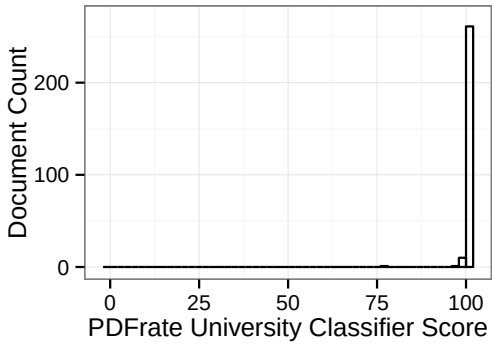
(a) Benign Documents, Contagio Classifier



(b) Malicious Documents, Contagio Classifier



(c) Benign Documents, University Classifier



(d) Malicious Documents, University Classifier

Figure 5.3: Operational Evaluation Score Distributions

with the Contagio classifier and added samples from the operational training set.

Using the original Contagio training data set, I determined the rating of all the observations in the operational training set. In an operational setting, all observations above the uncertain threshold (scores greater than 25) would typically require additional investigation, whether the outcome is uncertain or malicious. There were 200 documents in the operational training set matching this criteria. Of these 200 samples, 43 would be false positives and 14 would be false negatives using a traditional threshold. I added these 200 observations to the Contagio training set with the correct ground truth and created another classifier.

For comparison, I also created additional classifiers with varying sized randomly selected

Table 5.8: Scores of Benign Documents from Operational Evaluation Set Using Contagio Classifier Supplemented with Operational Training Data

		Benign		Malicious	
Additional Training Data	Training Set Size		Uncertain		
None (original Contagio)	10000	98076	1408	203	40
Random subset 2500	12500	99332	265	98	32
Random subset 5000	15000	99444	200	71	12
Random subset 7500	17500	99502	169	49	7
Uncertain and Malicious	10200	99506	183	26	12
Full training partition	20000	99540	134	48	5

Table 5.9: Scores of Malicious Documents from Operational Evaluation Set Using Contagio Classifier Supplemented with Operational Training Data

		Benign		Malicious	
Additional Training Data	Training Set Size		Uncertain		
None (original Contagio)	10000	0	0	19	254
Random subset 2500	12500	0	14	4	255
Random subset 5000	15000	0	14	4	255
Random subset 7500	17500	0	14	4	255
Uncertain and Malicious	10200	0	14	7	252
Full training partition	20000	0	14	4	255

subsets of the operational training set to simulate randomly selected additions to the Contagio classifier. The performance of these classifiers applied to the operational evaluation set is demonstrated in Table 5.8 and Table 5.9.

These results indicate that local tuning of the classifier has a great effect on improving the accuracy of the classifier. Note that shifting a few samples across the score midpoint in the wrong direction, as occurs with the malicious observations, is not considered harmful as these samples are already deep in the uncertain range (very close to the 50% vote mark) as shown in Figure 5.3b. The ratio of observations in the benign region (certain true negatives) rises from 98.3% to 99.8% for either of the top two re-training strategies, even surpassing the accuracy of the generally superior University classifier (99.5%). The corresponding drop in false positives is important because it coincides with a drop in uncertain observations.

In this case, if an operator responds to all uncertain or malicious observations, the majority of alerts will be true positives.

The random subset training additions have the outcome anticipated by intuition. As the number of random samples added from the training set increases, the classification results on the partitioned evaluation data improve. Adding the samples above the uncertain threshold from the training partition results in a classifier that is very close in accuracy to that constructed with the complete training partition. It follows that mutual agreement analysis is effective at identifying the observations on which the classifier performs poorly. It also follows that adding these samples to the training set does indeed improve the classifier by providing support in the region near these samples. On the other hand, adding the observations for which there is high mutual agreement improves the classifier very little. The result of adding the whole training set and adding the uncertain samples is similar, but the effort invested is drastically different. The difference in obtaining ground truth and adding 10,000 vs. 200 observations to the training set is monumental.

5.5 Evaluation on Virustotal Data

I measured the mutual agreement of PDFrate scores for Virustotal submissions during the year following the latest re-training of the University classifier, which occured in October 2013. From a corpus of PDF documents organized by initial upload to Virustotal, I randomly select 500 benign and 500 malicious documents per month. I consider any sample that has a detection by 3 or more AV engines as malicious and any that has less as benign.

Table 5.10 contains the two PDFrate classifier outcomes for the malicious samples, and Table 5.11 for the benign samples. I present monthly results for the University classifier, but for brevity, only present the year total for the Contagio classifier. These tables present the number of documents that receive classifier ratings of benign, uncertain, or malicious. I keep the convention of showing the split in the middle of the uncertain region based on a 50% score, allowing better comparison to standard classifier predictions and better showing the distribution of the scores. Generally, these tables demonstrate that the classifiers cast

Table 5.10: Outcomes for Benign Documents from VirusTotal

University Classifier

	Benign		Malicious	
Date		Uncertain		
201311	7	4	11	478
201312	2	0	2	496
201401	2	1	20	477
201402	10	6	16	468
201403	2	20	19	459
201404	9	10	19	462
201405	3	4	4	489
201406	20	9	22	449
201407	11	2	8	479
201408	20	18	22	440
201409	2	25	14	459
201410	7	21	5	467
total	95	120	162	5623

Contagio Classifier

total	841	1246	667	3246

the majority of their votes for the correct class, malicious and benign respectively. The counts drop off rapidly through the uncertain outcomes, and the incorrect class is a rare outcome. The distribution of ensemble classifier voting scores for the University classifier is shown in Figure 5.4.

The primary observation is that using mutual agreement to add an additional outcome or prediction of uncertain dramatically decreases classifier error. This comes at the expense of a small number of observations receiving a prediction of uncertain. Table 5.12 compares the predictions of a traditional classifier with that using mutual agreement analysis. Classification error and uncertain rates are presented. For the University classifier, the false positive rate (FPR) drops from 0.22% to 0.08% and the false negative rate (FNR) drops from 3.57% to 1.58%. The trade-off is that 3.68% of the incoming observations are classified as uncertain. Of the observations labeled uncertain, 34% would be misclassifications with a traditional vote threshold. For the Contagio classifier, 54% of the uncertains would be

Table 5.11: Outcomes for Benign Documents from VirusTotal

University Classifier

	Benign		Malicious	
Date		Uncertain		
201311	479	19	0	2
201312	494	5	1	0
201401	483	14	3	0
201402	480	19	1	0
201403	493	6	1	0
201404	492	5	2	1
201405	490	9	0	1
201406	483	17	0	0
201407	485	14	0	1
201408	482	18	0	0
201409	491	9	0	0
201410	483	17	0	0
total	5835	152	8	5

Contagio Classifier

total	5638	280	72	10

Table 5.12: Comparison of Classifier Performance Using Conventional Vote Threshold and Mutual Agreement Derived Uncertain Rate (UR).

University Classifier

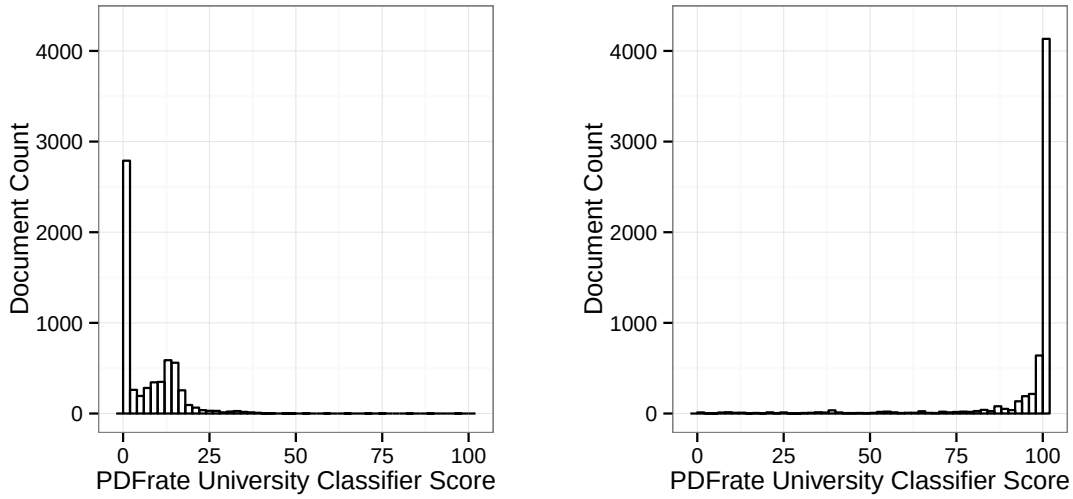
	FPR	FNR	UR
Conventional	0.22%	3.57%	-
Mutual Agreement	0.08%	1.58%	3.68%

Contagio Classifier

	FPR	FNR	UR
Conventional	1.37%	34.8%	-
Mutual Agreement	0.17%	14.0%	18.9%

classification errors. Note that I report the Uncertain Rate (UR) using the count of all observations as the denominator, while I use the conventional definition for FPR and FNR which use the count of the benign and malicious observations as the denominator.

One would typically expect the portion of classifier errors in the uncertain outcome to



(a) VirusTotal Benign Documents

(b) VirusTotal Malicious Documents

Figure 5.4: VirusTotal Score Distributions, University Classifier

be under 50%, at least when counts of benign and malcious samples are equal. Even the most uncertain classifier, a random guess, should yield the correct prediction half the time. So even uncertain predictions should be correct about 50% of the time. Hence, 50% should be the normal upper bound for the classification error rate inside the uncertain outcome. The Contagio classifier exceeds this slightly because of it has an irregular score distribution.

With the known classes of the samples labeled, it is clear that the University classifier is superior to the Contagio classifier. This is expected, as the Contagio classifier contains over an order of magnitude fewer documents and was compiled nearly three years before the University classifier. Without any external knowledge, the mutual agreement analysis derived Uncertain Rates of 3.68% for the University classifier and 18.9% for the Contagio classifier gives us an objective measure of the relative confidence of these classifiers. These measures are very close to the ground truth misclassification rates of 1.9% and 18.1% respectively. The ability to estimate classifier error with no knowledge of ground truth makes the mutual analysis derived Uncertain Rate extremely valuable.

One surprising observation from this data is the lack of a steep decrease in classification

accuracy throughout the year following the training of the classifier. It might be anticipated that the classifier would need to be retrained frequently to retain accuracy. I suspect that the low drift in the malicious documents over time was due to a lack of active evasion attempts against PDFrate. While polymorphism may be used to attempt to defeat signatures, rapid changes to the features used by PDFrate do not appear to occur in Virustotal submissions. I also tried to correlate exploits over time with classifier error and could discern no strong correlations between new software vulnerabilities and classifier evasion. In fact, the most common exploit found in the samples labeled uncertain was CVE-2010-0188, a very old, if not prolific, exploit. This exploit was the most common exploit reported in my VirusTotal submission data set. My labeling of exploits was limited to the analysis provided by the cumulative detections of the AV engines in VirusTotal, which may introduce a bias in this analysis. To the degree my ability to correctly identify the exploits used in documents was not biased, it appears that new exploits are not associated with PDFrate evasion. It also appears that the various techniques used to defeat signature matching are generally orthogonal to the attributes that PDFrate uses for classification. This implies that PDFrate and signature matching techniques complement each other well.

It is also noteworthy that the false negative rate is higher than the false positive rate and the contribution to the uncertain outcome is also higher from the malicious samples than the benign samples. This has a few implications. First, it seems that the classification PDFrate provides is more volatile for malicious samples than for benign–possibly due to less variation in benign samples. I presented equal quantities of benign and malicious documents, but most environments are heavily skewed to benign observations. Hence, classification error and uncertain rates will drop in a typical, mostly benign, environment.

Despite covering half of the possible voting score range (using a 50% mutual agreement threshold), the uncertain result occurs relatively infrequently in practice because the bulk of the scores reside at the ends of the spectrum. Removing the observations with high ensemble classifier disagreement allows the classification error to drop dramatically. Mutual agreement analysis permits a higher degree of confidence in the outcome of a classifier

without additional external information.

5.6 Independent Evasion Attack Evaluations

I evaluated the degree to which mutual agreement analysis allows identification of classifier evasion using published evasion studies against PDFrate: Mimicus [107], EvadeML [120], Reverse Mimicry [70, 71], and Parser Confusion [23].

5.6.1 Mimicus Evaluation

To demonstrate the utility of mutual agreement analysis in identifying observations that evade detection, I reproduced the work of Šrndić and Laskov [107] and applied mutual agreement analysis to these evasion attempts. I used the Mimicus framework to generate PDF documents that implement various evasion attack scenarios. I used the same data sets as the Šrndić and Laskov publication and submitted the resulting documents to pdfrate.com to obtain scores. Because I used the same attack data, my results are limited to 100 samples per attack type. I was able to achieve results that closely mirrored those documented in the Mimicus study.

I present the results of classification using mutual agreement from the various attack scenarios in Table 5.13. Note that since all of these documents are malicious, the correct classification is malicious. A rating of benign indicates successful evasion.

The distribution of PDFrate voting scores for the documents in each non-GD-KDE scenario is demonstrated in Figure 5.5. The GD-KDE attacks will be addressed specifically in Section 5.8. The vote score distribution of these attacks is largely disjoint of that in typical benign or malicious observations. Using an ensemble classifier diversity based approach, the majority of these attacks can be separated from benign observations. Hence, these attacks should be considered weak mimicry attempts.

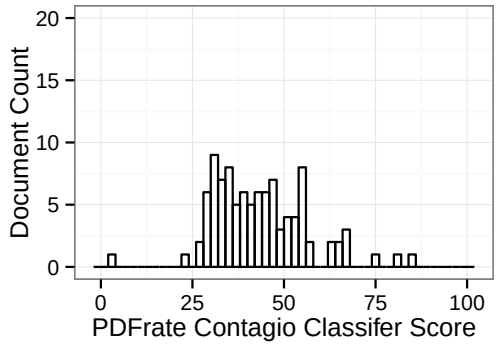
When all attributes of the classifier are known, 33% of the attacks are effective. However, when either the details of the classifier or the training set are withheld, the attack success

Table 5.13: PDFrate Contagio Classifier Outcomes for Mimicus Evasion Attacks

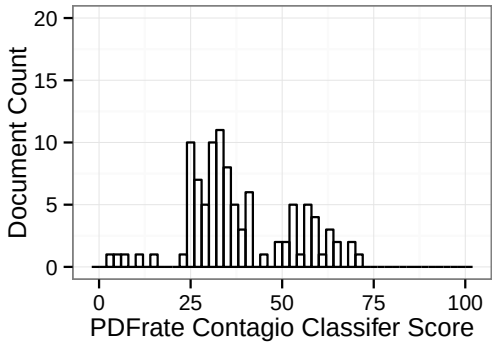
	Benign		Malicious	
Scenario		Uncertain		
Baseline Attack	0	0	0	100
F_mimicry	2	70	26	2
FC_mimicry	7	78	15	0
FT_mimicry	10	64	26	0
FTC_mimicry	33	62	5	0
F_gdkde	7	92	1	0
FT_gdkde	4	95	0	1

rate drops to 10% or lower. In addition to evaluation against the Contagio data set, the mimicry attack data was tested against the classifier trained with the University data set. This results in an alternate FC attack scenario because the training set is unknown to the attacker. Figure 5.6 shows the distribution of scores from applying the malware from the FTC attack scenario against the Contagio classifier to the University classifier. The results are very similar between the two classifiers. In both cases, only seven of the 100 evasion attempts are classified as benign. Carefully comparing Figure 5.5c and Figure 5.6 yields the observation that the University classifier provides a tighter cluster of scores near the center of the disagreement region. The results from the Contagio classifier are similar to that of the University classifier because the Mimicus evasion attempts use Contagio data for both baseline benign and attack data.

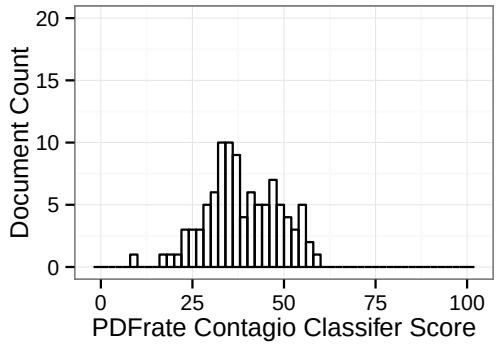
When mutual agreement analysis is employed, the majority of mimicus attacks are labeled as uncertain, indicating known classifier failure and possible evasion. In the best mimicry attack scenario, where all attributes of PDFrate are known, only 33% of the mimicry attempts are successfully classified as benign. If some details of the classifier, such as the exact training set, are not known by the attacker, then the mimicry success rate is below 10%.



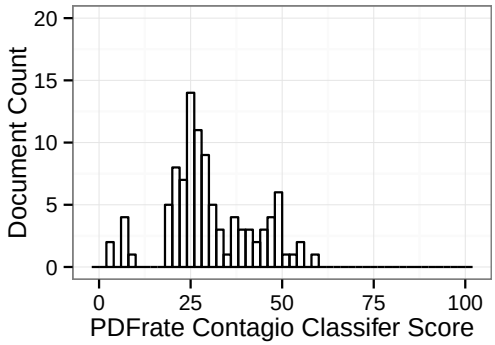
(a) Score Distribution for F_Mimicry Attack



(b) Score Distribution for FT_Mimicry Attack



(c) Score Distribution for FC_Mimicry Attack



(d) Score Distribution for FTC_Mimicry Attack

Figure 5.5: Mimicus Score Distributions, Contagio Classifier

5.6.2 EvadeML Evaluation

I evaluated the effectiveness of mutual agreement analysis applied to the EvadeML attacks advanced by Xu et al. [120]. The procedures to replicate the EvadeML attacks are publicly documented, including source code [3]. In addition, Xu et al. provided the documents used in their published evaluation for validation against PDFrate.

The evaluation performed by Xu et al. resulted in 16,985 documents derived from 500 seed malicious PDFs. Unlike the Mimicus and Reverse Mimicry attacks, there is only one evasion scenario advanced in EvadeML. Since there are multiple samples derived from a single seed, I provide results for all document mutations and the best mutation (lowest

[3] http://evademl.org

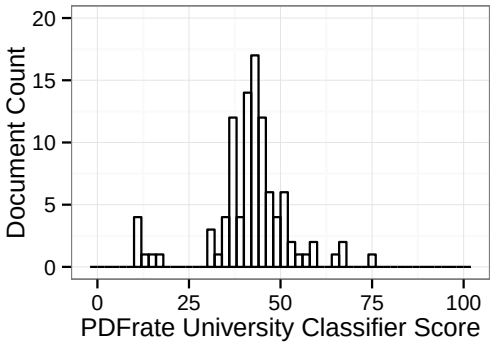


Figure 5.6: Score Distribution for FC_Mimicry Attack, University Classifier

Table 5.14: PDFrate Outcomes for EvadeML Attacks

Contagio Classifier

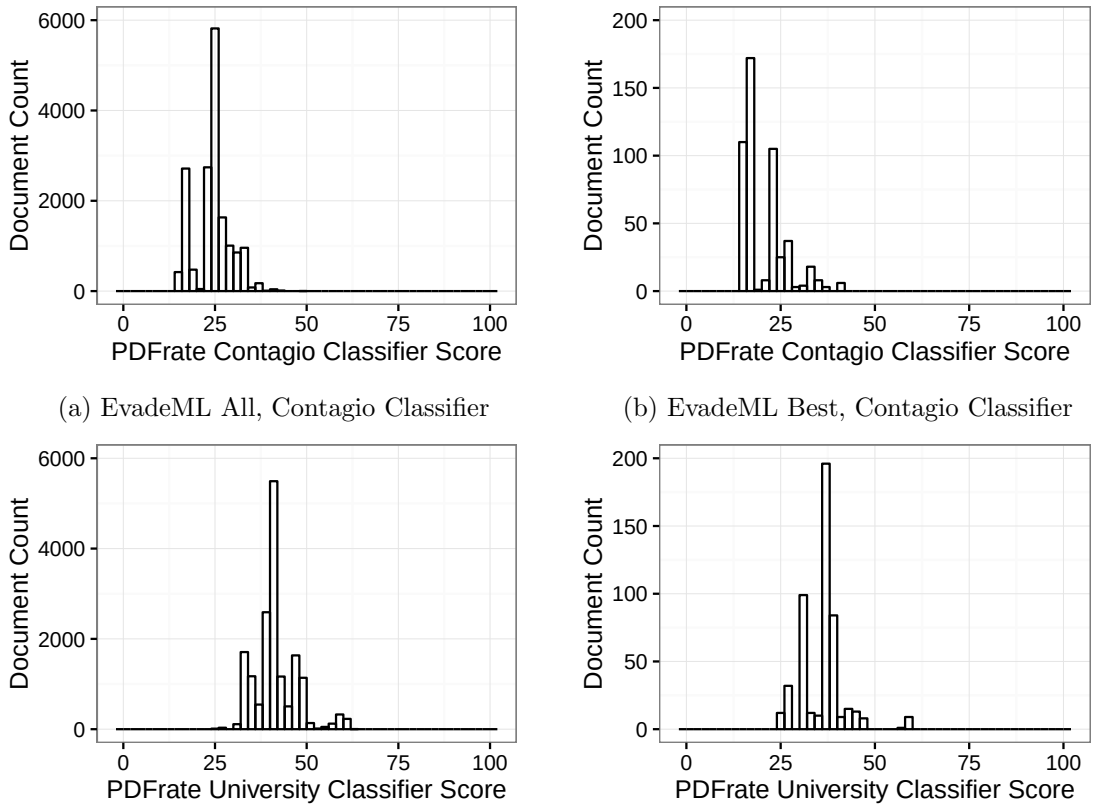
	Benign		Malicious	
Scenario		Uncertain		
All	57.5	42.5	0.0	0.0
Best	81.8	18.2	0.0	0.0

University Classifier

	Benign		Malicious	
Scenario		Uncertain		
All	0.0	94.8	5.2	0.0
Best	0.8	97.2	2.0	0.0

PDFrate score) for each seed document. Table 5.14 contains the results for the application of mutual agreement analysis to the EvadeML attacks using both the Contagio and University classifiers. To remain comparable with the results from the Mimicus and Reverse Mimicry attacks, I report these results as a percentage of all the documents for their respective attacks sets. Figure 5.7 depicts the score distributions for these various evasion attacks.

Analyzing mutual agreement in the PDFrate outcomes for the EvadeML results in different outcomes based on the training set. Many attacks against the Contagio classifier are effective, falling below the 50% mutual agreement threshold or below a 25% voting score. In fact, over 80% of the best attacks per seed fall below the 25% vote score threshold, making



(a) EvadeML All, Contagio Classifier

(b) EvadeML Best, Contagio Classifier

(c) EvadeML All, University Classifier

(d) EvadeML Best, University Classifier

Figure 5.7: EvadeML Score Distributions

this a relatively strong attack. However, less than 1% of the attacks are effective against the University classifier.

The EvadeML attacks used the Contagio data set for evaluation during construction. It is difficult to determine how much of the difference in the score distributions for the two classifiers is due to general differences in quality (University classifier contains many more training samples) and how much is due to the University classifier being unknown to the attacker. The large difference in the distributions for the two classifiers and similar qualitative results when compared to that of Reverse Mimicry attacks seem to indicate that the quality of the training sets is important in preventing evasion attacks. On the other hand, the difference in score distributions for all the mutations and the best mutations indicate

that there is a large degree of variance in the outcomes using the genetic programming approach. Indeed, Xu et al. observed that the longest traces were not necessarily the most evasive. Hence, optimizing these attacks, or finding the best attack for a given classifier, probably requires testing against that classifier. As with the other attacks, it appears that a private training set does improve evasion resistance.

The EvadeML attacks are able to attain over 80% evasion when the classifier based on the publicly known Contagio data set is used. However, when the University classifier, which higher quality and unknown to the attacker, is used, under 1% of evasion attacks are successful.

5.6.3 Reverse Mimicry Evaluation

I also applied mutual agreement analysis to the Reverse Mimicry attack proposed by Maiorca et al. [70, 71]. The exact procedures required to replicate these attacks are not publicly documented. However, Maiorca et al. provided me with the documents used in their studies. Their most recent attacks involved approximately 500 documents in each evasion scenario. To remain consistent with the Mimicus attack evaluation, I took a 100 sample random subset of each evasion attack for my evaluation.

In Table 5.15, I present the results of applying mutual agreement analysis to the Reverse Mimicry attacks against both the Contagio and University classifiers. The score distributions for these attacks against the University classifier are shown in Figure 5.8 In spite of mutual agreement analysis, 67% of the Reverse Mimicry attacks are successful evasions (considered benign) against the Contagio classifier.

The University classifier fares much better than the Contagio classifier. The only evasions against the University classifier are achieved by the PDFembed attack. This attack is so successful because a complete malicious PDF is embedded in an otherwise benign document. This embedded document resides in a compressed data stream, which means that the structural features cannot be observed by PDFrate's feature extractor. This is in contrast to the other scenarios, EXEembed and JSinject, where despite efforts at minimization, some

Table 5.15: PDFrate Outcomes For Reverse Mimicry Attacks

Contagio Classifier

	Benign		Malicious	
Scenario		Uncertain		
EXEembed	77	22	1	0
PDFembed	93	7	0	0
JSinject	30	67	3	0

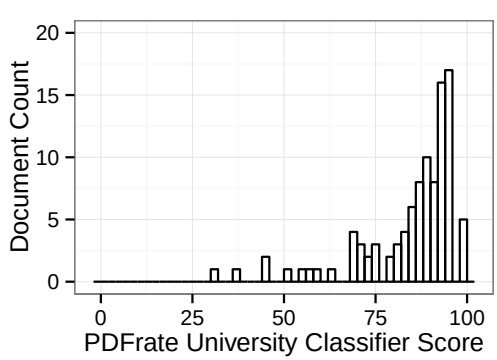
University Classifier

	Benign		Malicious	
Scenario		Uncertain		
EXEembed	0	4	16	80
PDFembed	81	19	0	0
JSinject	0	22	55	23

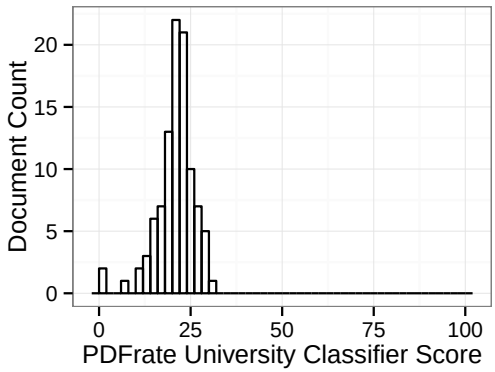
indicators of malfeasance remain exposed.

The PDFembed scenario is effective against the detector at pdfrate.com because it does not perform recursive decoding and analysis as would be necessary in an operational system. This failure is similar to malware analysis systems that assume an input of an unpacked executable and fail when presented with a packed executable or a Trojan document. When PDFrate is deployed in operational detection systems, it is usually done within a framework that provides both decoding of PDF streams and extraction of PDFs from other containers such as emails or zip files [8]. In all the PDFembed attacks, the embedded document was identical. The Contagio and University classifiers both easily detect this document with high confidence once it is extracted, returning scores of 97.6% and 100% respectively.

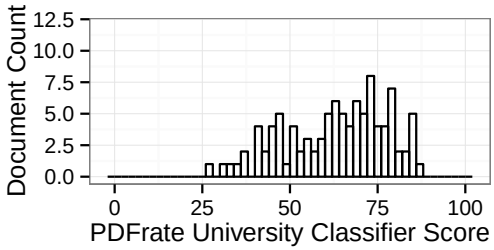
For the isolated PDFrate implementation, the PDFembed scenario represents a strong evasion scenario, where classifier introspection provides little benefit because the feature extractor is evaded so well. Even though the Contagio based classifier is a poor fit for the malware used in the EXEembed and JSinject, many of these samples still fall in the uncertain outcome vote range. When the stronger University classifier is used, mutual agreement analysis flags these evasion scenarios that would otherwise be successful.



(a) EXEembed Attack



(b) PDFembed Attack



(c) JSinject Attack

Figure 5.8: Reverse Mimicry Score Distributions, University Classifier

Table 5.16: PDFrate Scores for Parser Confusion Attacks

Attack Scenario	Contagio Classifier	University Classifier
Base Malicious	86.4%	89.6%
Parser Confusion	70.0%	65.8%
Parser Confusion and Reverse Mimicry	7.8%	2.3%

5.6.4 Parser Confusion Evaluation

We also evaluated the effectiveness of the parser confusion attacks of Carmony et al. A single base malicious document implementing their parser confusion and reverse mimicry attacks was tested against PDF in their evaluation. The hash of the document for each attack variant was published in their study. The PDFrate scores for these attack methods are shown in Figure 5.16.

The Parser Confusion attack is able to drop the voting score into the uncertain region, but it remains far from a successful evasion. This result is explained by the fact that PDFrate focuses on structural features instead of content, limiting the impact of this method. However, when combined with the reverse mimicry attack, the result is strong evasion. This is explained in the same manner as the PDFembed attack discussed in Section 5.6.3. This attack is effective because the features indicating the malicious activity are hidden from the PDFrate feature extractor through various layers of encoding and encryption. However, due to the parser confusion attacks, the file is not trivially decoded with existing PDF parsing libraries or tools. Hence, even if PDFrate is used in a framework that provides recursive PDF parsing and analysis, the parsing will not be effective if the parser is subverted.

5.7 Mutual Agreement Threshold Tuning

For most of my evaluations, I used a 50% mutual agreement threshold, which splits the classifier voting score region into four equal sized quadrants. It is possible to choose an arbitrary mutual agreement threshold. In Table 5.17, I present the PDFrate University classifier outcomes applied to the operational evaluation data set and the FC Mimicus attacks across the full mutual agreement range.

The exact mutual agreement threshold chosen strikes a balance between improvement in classification failure detection and the number of classifier predictions thrown out as uncertain. Operators who wish to have a lower amount of uncertain outcomes may choose a lower threshold. Taking the PDFrate performance in Table 5.17 as an example, if 30% is selected as a threshold, the uncertain region comprises ensemble classifier voting scores between 35% and 65% instead of 25% and 75% with a 50% threshold. For the operational data set, the uncertain rate for benign samples drops from 0.456% to 0.256%. However, the number of successful evasion attempts rises from 7% to 12%. The optimal setting for this threshold depends on the preferences of the operator. The sensitivity of uncertain detection is adjusted by tuning the mutual agreement threshold, setting the boundaries for the uncertain range.

Table 5.17: University Classifier Performance as Mutual Agreement Threshold Is Adjusted

Benign Operational Evaluation

Mutual Agreement Threshold	Uncertain Score Range	True Negative Rate (TNR)	False Positive Rate (FPR)	Uncertain Rate (UR)
0%	-	99.8%	0.150%	0%
10%	(45,55)	99.8%	0.128%	0.0592%
20%	(40,60)	99.7%	0.103%	0.147%
30%	(35,65)	99.7%	0.0832%	0.256%
40%	(30,70)	99.6%	0.0712%	0.342%
50%	(25,75)	99.5%	0.0552%	0.456%
60%	(20,80)	99.3%	0.0331%	0.618%
70%	(15,85)	99.1%	0.0291%	0.825%
80%	(10,90)	98.7%	0.0261%	1.27%
90%	(5,95)	97.0%	0.0120%	3.01%
100%	(0,100)	53.6%	0%	46.4%

Malicious Operational Evaluation

Threshold	Range	FNR	TPR	UR
0%	-	0%	100%	0%
10%	(45,55)	0%	100%	0%
50%	(25,75)	0%	100%	0%
60%	(20,80)	0%	99.6%	0.366%
70%	(15,85)	0%	99.6%	0.366%
80%	(10,90)	0%	99.6%	0.366%
90%	(5,95)	0%	99.6%	0.366%
100%	(0,100)	0%	95.6%	4.40%

Mimicus FC Attack

Threshold	Range	FNR	TPR	UR
0%	-	84%	16%	0%
10%	(45,55)	69%	8%	23%
20%	(40,60)	31%	4%	65%
30%	(35,65)	12%	4%	84%
40%	(30,70)	7%	1%	92%
50%	(25,75)	7%	1%	92%
60%	(20,80)	7%	0%	93%
70%	(15,85)	6%	0%	94%
80%	(10,90)	0%	0%	100%
90%	(5,95)	0%	0%	100%
100%	(0,100)	0%	0%	100%

5.8 Ensemble Classifier Diversity

Mutual agreement analysis should apply to all ensemble classifiers that provide sufficient diversity in individual classifiers. To validate this, I studied the feasibility of countering evasion against SVMs by applying mutual agreement analysis to SVMs using an ensemble approach.

The Mimicus attack framework implements a Gradient Descent and Kernel Density Estimation (GD-KDE) attack against their PDFrate replica utilizing an SVM classifier. This attack operates by exploiting the known decision boundary of a differentiable classifier [17].

I reproduced the GD-KDE evasion attacks of Mimicus and confirm that they are indeed extremely effective. Using the e1071 package of R [4], which relies on libSVM [24], I calculated the average probability of 8.9% malicious (or 91.1% benign) for both GD-KDE scenarios, putting these attacks squarely within the evasion region. Šrndić and Laskov use the scaled distance from the SVM decision boundary of a different SVM implementation to provide a similar result. The GD-KDE attacks demonstrate that introspection of a single classifier such as SVM cannot be relied upon to detect evasions.

While effective against an SVM classifier, the results on PDFrate's Random Forest classifier using the GD-KDE attack are roughly comparable to the conventional counterparts (see Table 5.13). It is not practical to wage a similar type of attack against Random Forests because Random Forests have extremely complex and stochastic decision boundaries.

I sought to determine the extent to which I could make an SVM classifier identify probable evasions through diversity enabled introspection. I implemented a simple SVM based ensemble classifier using 100 independent SVM classifiers with the score being the simple sum of the votes of individual classifiers. To determine the attributes important to building diversity in ensembles, I varied the subset of features and training data used in constructing each of the individual SVMs. I performed a full grid search. The most salient results are reported in Table 5.18, which shows feature bagging using the full training data set, and Table 5.19, which shows bagging on training data using the full feature set. These

[4] http://www.r-project.org/

Table 5.18: Number of Documents per GD-KDE Attack Where Ensemble SVM Classifier Provides Correct Prediction as Portion of Features Used Is Varied

	Feature Subset			
Attack	5%	7.5%	10%	12.5%
Baseline Malicious	100	99	98	98
Baseline Benign	2	41	93	94
F_gdkde	100	100	99	5
FT_gdkde	99	100	92	1

Table 5.19: Number of Documents per GD-KDE Attack Where Ensemble SVM Classifier Provides Correct Prediction as Portion of Training Data Used Is Varied

	Training Data Subset			
Attack	12.5%	25%	50%	100%
Baseline Malicious	86	87	92	98
Baseline Benign	100	100	100	100
F_gdkde	0	0	0	0
FT_gdkde	0	0	0	0

tables demonstrate the portion of classifier outcomes that match the correct result (desired result for evasion attempts is malicious or uncertain).

It appears that bagging of training data is not particularly important in building an ensemble classifier where mutual agreement analysis is useful. To my amazement, I found no situation where anything but the full training set provided the best results. However, bagging of features is critical to constructing a classifier where mutual agreement analysis is able to identify uncertain predictions. This bagging of features for an SVM classifier provides the necessary diversity in extrapolation that makes mutual agreement analysis meaningful. It seems that the individual classifiers based on subsets of the complete feature set are much harder to evade collectively than a single classifier using all the features. While a single classifier can be evaded by successfully mimicking a subset of the features, it appears that a combination of multiple classifiers based on a small number of features requires a more complete mimicry across the full feature set. The application of feature bagging to

Table 5.20: PDFrate SVM Ensemble Classifier Outcomes for GD-KDE Attacks

	Benign		Malicious	
Attack		Uncertain		
Baseline Malicious	0	0	2	98
Baseline Benign	93	7	0	0
F_gdkde	3	97	0	0
FT_gdkde	8	91	1	0

the many independent SVMs makes a GD-KDE style attack infeasible as there is no longer a single predictable decision boundary to attack.

The results also indicate that careful tuning of the portion of features used in bagging is critical when using an SVM based ensemble. There seems to be a trade-off between the ability to correctly classify malicious observations (including evasion attempts) by using fewer features in each classifier, and benign observations by using more features. The use of fewer features results in a more complex classifier with smaller divisions while more features moves closer to a standard SVM, which has a single hyperplane divider. This result might be explained by suggesting that the features used in PDFrate provide better extrapolation for benign samples but that malicious samples have higher variation in PDFrate's features requiring more similar training samples for successful classification.

Table 5.20 shows the outcomes of the SVM ensemble classifier applied to the Mimicus GD-KDE attacks and baseline benign and malicious samples. The outcome shows that while the evasion attempts are successful in dropping the scores out of the malicious range, the vast majority of the evasion attempts fall in the uncertain range. Only 8% of the evasion attempts are fully successful in the best scenario while only 4.5% of the known data is in the uncertain region. These results are comparable to results obtained using PDFrate's Random Forest classifier where GD-KDE attacks are not possible. Hence, mutual agreement analysis applies not only to Random Forests, but seems to apply generally to all ensembles that have adequate diversity. Bagging of features appears central to this capability.

5.9 Evaluation on Drebin Android Malware Detector

Ensemble classifier mutual agreement analysis should be applicable to all situations where evasion is possible, including other malware classifiers. I evaluated the utility of mutual agreement analysis on the Drebin Android malware detector [9]. Drebin is a strong complement to PDFrate because it operates on a software package instead of a document and utilizes many string based/binary features instead of numerical features. Since the data used in the original Drebin study has been published, I use this data for my evaluation.

Drebin operates by performing a quick scan to extract features from the Android application manifest and disassembled code. These features are formatted as strings. Features extracted from the manifest include the names/values of hardware components, requested permissions, application components, and intents (message framework). The values of API calls, used permissions, and network addresses/URLs are taken from the dissembled code. The string values are mapped into a binary feature vector containing over 500,000 unique values.

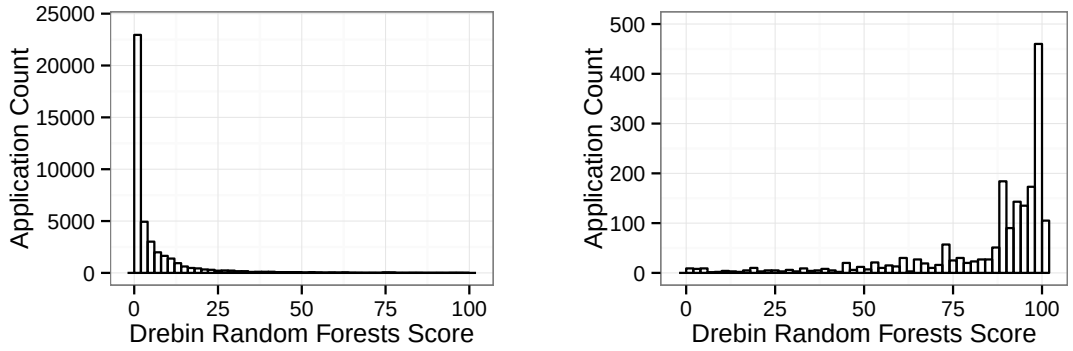
A linear SVM is trained offline and used to provide weights (distance from hyperplane) for each feature observed during classification. This per predictor weight is combined to provide an overall score and compared to a threshold to determine the outcome. Due to this scheme, Drebin provides a malicious score and can identify variables that contribute to this score. Drebin is evaluated with over 100,000 benign and 5,000 malicious samples, providing a false positive rate of 1% and a malware detection rate of nearly 94%.

Since the linear SVM's score is based on a single division, it does not have enough diversity to provide a measure of internal coherence. I employ a Random Forest classifier, which requires adapting the features to ensure computational efficiency and to ensure results comparable to the original linear SVM. Instead of using all string values as features, I use a subset of 891 features that comprise the most resilient features. I use all of the features for constrained categories such as API calls and permissions. For arbitrarily named attributes, such as components and intents, I use the most prolific values, selecting those which occur over 100 times in the training set. I ignore specific values for highly volatile items such as

URLs and network addresses, which compose over half the features used by Drebin. Lastly, I sum the occurrences of each category of features and use these counts as features. As an optimization, I de-duplicated any equivalent feature vectors during classifier training (not during evaluation). This de-duplication using our narrow feature set results in a reduction from 123,453 to 63,379 unique benign and 5,560 to 2,185 unique malicious samples. Barring these transformations of data, I used the published Drebin data sets and data set partitions in my evaluation.

I tuned my Random Forest based classifier to provide classification performance comparable to the linear SVM classifier of Drebin. The primary item I tuned was the ratio of benign to malicious samples used in training each tree. This was necessary because there is an extreme imbalance in the benign to malicious ratio of the various training sets. I tuned this ratio of benign to malicious for individual tree training to 2.5 benign to 1 malicious in order to match the desired false positive rate of 1% chosen by Arp et al. I set the other tunable parameters for Random Forest to standard values: each Random Forest contained 1000 trees and number of variables tried at each split was set to the square root of the number of features. My Random Forest classifier provided an average false positive rate of 1.06% and a malware detection rate of 92.3% on the published data set partitions using a traditional threshold without an uncertain region. The Random Forest based classifier performance is very similar, albeit slightly inferior to that provided by Drebin's linear SVM. Figure 5.9a shows the distribution of scores for the benign samples using one of the published data set partitions. Figure 5.9b shows the same for the malicious samples. As expected, the score distributions are shaped similarly to that of PDFrate, but since the classifier accuracy is lower, the samples are distributed farther from the respective ends of the score continuum. Table 5.21 shows the classifier outcomes for typical mutual agreement thresholds.

An important facet of the original Drebin study is the division of the malware by family and evaluation of the classifier on previously unknown malware families. This was achieved by withholding the family to be evaluated from the training set, and then applying the resultant classifier to the malware samples in that family. It is noted by Arp et al. that



(a) Benign Samples

(b) Malicious Samples

Figure 5.9: Score Distribution Random Forests Based Drebin Classifier

Table 5.21: Drebin Random Forest Outcomes as Mutual Agreement Threshold Is Adjusted

Benign Samples

	Benign (%)		Malicious (%)	
Mutual Agreement Threshold (%)		Uncertain		
30	97.46	1.49	0.54	0.52
40	96.49	2.45	0.63	0.43
50	95.12	3.82	0.71	0.35

Malicious Samples

30	4.44	3.27	5.44	86.85
40	3.77	3.93	7.30	84.99
50	3.16	4.56	10.34	81.95

Drebin provides relatively poor classification of previously unknown malware. I applied my Random Forest based classifier and uncertain score region to this same problem. Figure 5.10 compares the detection rates of the linear SVM classifier and our Random Forest based classifier using mutual analysis agreement.

As expected, the vast majority of unknown malware families have the score distribution of a weak evasion attack, indicating that the classifier considers these observations similar to neither the benign or malicious samples seen in the training set. As an example, the scores of malware family A are shown in Figure 5.11. On average, 75.2% of every family

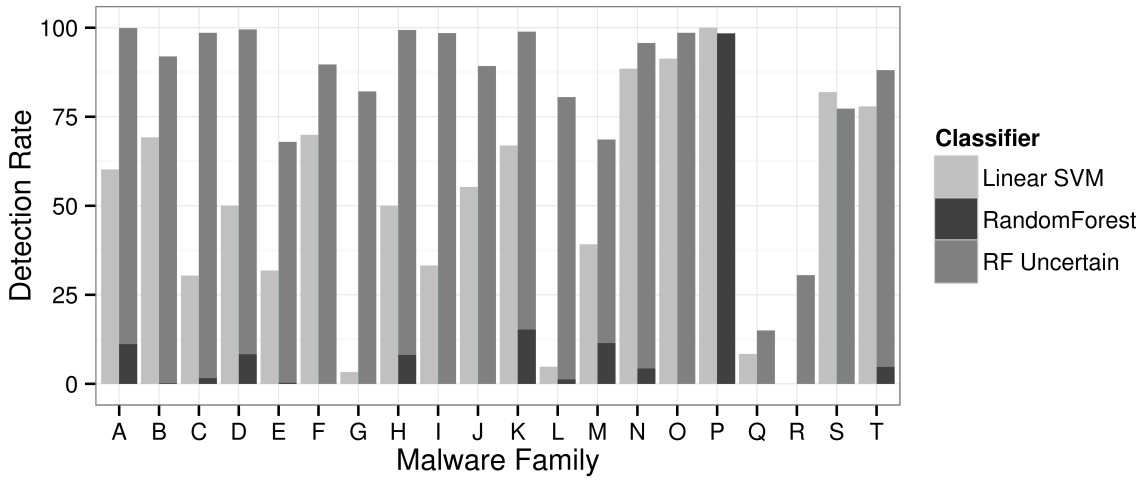


Figure 5.10: Comparison of Detection Rates for Previously Unknown Malware Families

is labeled as unknown and an additional 8.2% are labeled as malicious using our Random Forest base classifier, while 50.6% of every family is labeled as malicious by the Drebin linear SVM. Families Q and R represent strong evasion. Arp et al. note that Family R cannot be reliably detected with the feature set used by Drebin. While the features used by Drebin are sufficient for the detection of Family Q when included in the training set, it is too different from other families in Drebin's feature space to be flagged as an evasion. On the other hand, Family P is so similar to other malware families in Drebin's feature space, that it is not necessary to have samples of this family in the training set. Removing these three families, an average 89.7% of the samples in the remaining 17 families are identified as malicious or uncertain by the Random Forests while 53.2% are detected by the linear SVM. It should be considered advantageous to label these previously unknown samples as uncertain so that the operator can take action to improve the classifier. While the linear SVM provides the average classification accuracy of a coin toss in these scenarios, the mutual agreement conscious ensemble is able to flag the majority of the novel attacks as possible evasions.

Mutual agreement analysis does apply well to the Drebin Android malware detector. While the linear SVM classifier performs poorly on novel malware attacks, a Random Forest

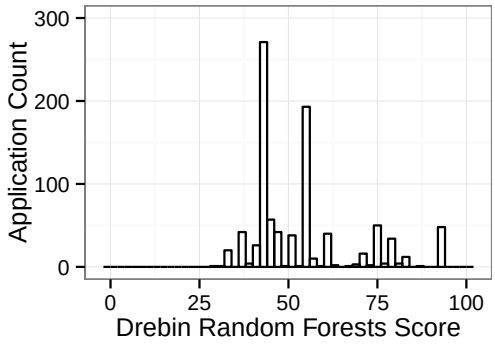


Figure 5.11: Score Distribution for Unknown Family A

classifier monitoring ensemble disagreement flags the majority of novel attacks as uncertain.

5.10 Discussion

Mutual agreement analysis in ensemble classifiers provides an estimate of confidence that the classifier prediction is accurate, without external validation. Many classifiers can provide a score continuum, such as the distance from the decision boundary used in SVM, but these metrics are not accurate in the face of mimicry attacks. Furthermore, conventional measures of confidence are not applicable to data that diverges from the population for which ground truth is known.

Mutual agreement reflects the internal consistency of the classifier. This internal consistency is a proxy for the confidence of the classifier, assuming adequate strength of the features. The attacks against PDFrate demonstrate that mimicry resistant features are critical to identification of novel attacks. If the feature extractor is resistant to tampering and the features are proper indicators of malfeasance, then novel attacks will either be detected as malicious or be rated as uncertain. However, if the feature set (or feature extraction mechanism) is weak, then evasion will still be possible. Operators must be vigilant to prevent evasion during the feature extraction phase of malware detection.

The evasion attacks that were successful against PDFrate, fully evaded PDFrate's features through embedding a malicious PDF in another, making the malicious PDF invisible to the feature extractor. Other attacks, while seeking to fool the feature extractor, were insufficient because some features were still operative. This evasion could be overcome by a parser that more fully mirrors that of the target application. However, this is difficult in practice due to the issues exposed by the Parser Confusion attacks. As an alternative to using the target reader program to extract features, one could add parsing errors or anomalous constructs to the feature set.

In building an ensemble using base SVM classifiers, I found feature bagging to be critical to generating the diversity necessary to make mutual agreement measurements meaningful. Unqualified, bagging refers to the utilization of random subsets of training data. This method is used extensively in machine learning techniques. In our study, bagging of training data was not shown to be important for mutual agreement analysis. This may have been due to a lack of diversity in that training set. Further studies might show under what conditions training data bagging provides diversity useful for facilitating mutual agreement analysis. I also observed that tuning the portion of features used in our SVM ensemble was important. I observed no similar need to tune the parameters of Random Forest, but this could be an area of future study. The number of features tried at each node (mtry) and the depth of the trees might impact the useful diversity in a Random Forest. It appears that many features, even if they are interdependent or have low classification value, contribute to make evasion more difficult.

If the features are strong, then the relevance of the training set will dictate the mutual agreement rating for individual observations. If the test observations are similar to samples in the training set, then high certainty predictions will occur. The test observations that differ from the training set in feature space will be given classifications considered uncertain by the classifier. In some instances in our evaluation, the quality of the training set was shown to be important to detection of evasion attacks. For example, the superior PDFrate University classifier had considerably fewer evasions than the Contagio classifier for the

Reverse Mimicry attacks. For the Drebin evaluation, Family R represented strong evasion due to weak features, but Family Q was detectable if it was included in the training set. Therefore, the effectiveness of mutual agreement analysis is also dependent upon adequate coverage in the training set. However, the effectiveness of the training set is directly dependent upon the strength of the features. A weak feature set will require a more expansive training set than a feature set that more closely models fundamental malicious attributes. Operators should ensure that features used for malware detection are not only resistant to spoofing but that they are based on artifacts caused by malware and not merely coincidental with current attacks.

As was shown in Section 5.6.1, keeping the training set of a classifier secret helps improves resiliency against targeted evasion attempts. It might be advisable for operational systems to hide the exact scores returned from their classifiers as these scores assist attackers in knowing if changes they make hurt or help their evasion attempts. This information could weaken the benefit provided by a secret training set [52].

The GD-KDE attacks of Mimicus demonstrate that some classifiers can make machine learning based detectors susceptible to evasion attacks. Stochastically generated ensemble classifiers have not been shown to be vulnerable to similar attacks, but new approaches might be found. The ability to measure mutual agreement in ensemble classifiers comes at little cost, but provides for detection of practical classifier evasion. This capability is a strong reason to use ensembles in situations where classifier evasion is a concern, such as in malware detectors. If mutual agreement analysis is used to optimize classifier training, then an attacker may have more knowledge of additions to the training set than if random selection is used. However, it is not obvious how this knowledge could be exploited by an attacker. Any effective attacks that use knowledge of mutual agreement based training optimization to poison the classifier would be important.

Some advocate the use of simple, monolithic classifiers, because the result is perceived as easier to interpret. For example, the ability of Drebin to identify the features that contribute to the classification is lauded. It is not clear, however, if this information is really useful to

end users. Users are already given the opportunity to review permissions and often choose incorrectly when prompted. Given that URLs and API calls can be socially engineered and that users are generally not aware of these elements, it is not likely that providing these items as context to a user will help them make a correct decision. For security professionals, ensemble classifiers provide mechanisms that aid in analysis such as similarity to existing known malicious or benign samples. Most importantly, the feature set will be useful to a trained analyst.

Mutual agreement analysis gives operators greater confidence in the accuracy of the classifier and the ability to prioritize response to alerts. Some operators will use ensemble classifier introspection simply to adjust the voting threshold. Environments that seek to avoid false negatives (evasion attacks) will use a low threshold and increase the number of false positives. On the other hand, some environments might use a higher than normal voting threshold to achieve a low false positive but potentially higher false negative rate, such as that achieved by antivirus engines. The operator gets the most benefit from mutual agreement analysis when uncertain observations are subjected to focused analysis. These samples must necessarily be subjected to different and complementary analysis or detections. Since the number of uncertain observations is low for a well performing classifier, this second opinion can be relatively expensive, possibly manually driven or involving dynamic analysis. Ensemble diversity based confidence estimates are useful for organizations that desire to identify novel attacks to perform additional analysis. While possibly unconventional for the machine learning field, the addition of the uncertain outcome is intuitive for the security field where many systems provide only adjudications for known observations, whether benign or malicious. For example, it is common for SPAM filters to utilize a quarantine for samples that cannot be classified reliably. Very often, high fidelity alerts are preferred over a response for every observation.

Mutual agreement analysis is very effective at identifying those samples that are not similar to the already known samples in the training set. Since adding uncertain samples to the classifier dramatically improves the classifier accuracy, analysis of uncertain observations

is likely to motivate rather than desensitize operators. Operators are empowered to improve the classifier in a manner much more effective than random additions to the training set.

Evaluation of machine learning based detectors might be improved though application of mutual agreement analysis. A concise metric is the Uncertain Rate, or portion of observations for which a classifier is poorly suited to provide a prediction. The effectiveness of classifier evaluation using the mutual agreement score distribution and variance could be a topic of future studies. The classifier score distributions shown in Figure 5.9a and Figure 5.9b seem to indicate that regression could be used to predict the amount of successful evasions. The difficulty in this type of analysis, however, is separating the arcs for the benign and malicious data when external ground truth is not provided.

Most importantly, monitoring mutual agreement in ensemble classifiers raises the bar for evasion, for both previously unseen attacks and targeted mimicry attacks. Contemporary evasion attacks, which have called into question the resiliency of learning based detectors, are shown to be weaker than previously supposed. Merely obfuscating attacks such that they no longer appear as known attacks is not enough. Successful mimicries must very closely mirror benign samples. Of course, future research into the degree to which mutual agreement analysis can improve attack quality is imperative.

Chapter 6: Specific Value Based Features

PDFrate relies on features that parameterize the documents. In all cases, these features quantify general metrics of the document, minimizing reliance on specific values or terms. For example, features such as the number of characters in a metadata field are used.

I studied the feasibility of using specific value or term based features in addition to general structural and metadata based features. Since these features are optional and easily modified by the document author, it was recognized these features were inadequate on their own. However, I sought to determine the degree to which employing observation of specific values could improve classification performance. I attempted various methods of turning these values into numeric features suitable for machine learning. I found that if bagging is used in a prevalence database, it is possible to train a classifier that is able to use the overlap in specific values when present without a large increase in false negatives when values do not overlap.

6.1 PDF Metadata Values

I studied the specific values found in PDF documents. I used the same regular expression based metadata and structural information based extractor. However, instead of deriving numeric features such as the number of objects, I operated on this data as strings or terms. This data can be categorized as follows:

- Author, Company, etc.: Metadata items tied to the author of the PDF.
- Creator, Producer, etc.: Metadata items likely that identify the tool used to generate the PDF.
- Title, Subject, etc.: Metadata items associated with the document content.

- Box dimensions, Image dimensions, etc.: Structural artifacts that reflect the layout of the document.

- CreateDate, InstanceID, etc.: Metadata items that should be unique on a per-document basis.

I studied the metadata for nearly 7 million PDFs taken from the same source as the operational data set. I observed that the opportunistic malicious PDFs had a much smaller amount of metadata than the other classes. It also appeared that many of the targeted malicious PDFs had consistent metadata. As examples of the types of metadata encountered, I present a selection of values from the Creator metadata item and layout box dimensions. I report the prevalence of a few specific values giving the count of the occurrences of these metadata values (can be more than once per document). I observed notable differences between the targeted attacks and the other document types (benign or opportunistic malicious) so I provide counts based on this division.

The Creator metadata values indicate the software used to generate the document. Table 6.1 shows some of the most common specific values that are exclusive to each class, as well as some that occur in both.

Table 6.1: Example Creator Value Frequencies

Targeted Malicious	Benign and Opportunistic Malicious	Value
22	0	
20	0	...A.c.r.o.b.a.t. ....Vh. .8...0
6	0	Advanced PDF Repair: http://www.pdf-repair.com
8	6769	Adobe LiveCycle Designer ES 8.2
5	724	...W.P.S. .O.f.f.i.c.e. N*N.rH
4	33	llPDFLib program
4	308	...A.c.r.o.b.a.t. .P.D.F.M.a.k.e.r. .9...0. .W.o.r.d. rH
0	298307	Microsoft Word
0	384345	PScript5.dll Version 5.2.2

The Box structural items reflect the dimensions of various types of boxes used in layout of the document. Table 6.2 shows some of the most common values for these classes. It should be noted that some values are either exclusively found within or without the targeted malicious class, while some are found in the benign class also.

Table 6.2: Example Box Value Frequencies

Targeted Malicious	Benign and Opportunistic Malicious	Value
24	0	Box: 8240x1 (other)
1	0	Box: 196x554 (other)
1	0	Box: 1691x1532 (other)
543	10362262	Box: 595x842 (A4)
242	95145391	Box: 612x792 (letter)
82	62	Box: 379x698 (other)
76	25	Box: 674x799 (other)
65	5108	Box: 14x20 (other)
40	151123	Box: 1000x1000 (other)
35	852932	Box: 2568x1314 (other)
34	5247	Box: 7x17 (other)
31	14447	Box: 1300x1135 (other)
31	19632217	Box: 0x0 (other)
30	44	Box: 81x4 (other)
0	6696393	Box: 1x0 (other)
0	8455450	Box: 10x10 (other)
0	8890847	Box: 0x1 (other)
0	10740361	Box: 432x648 (other)

6.2 Metadata and Structural Based Signatures

An obvious mode of employing specific value indexes would be to generate signatures for values appearing in one class but not appearing in others. A quick scan of the data indicates that one could generate static signatures for a few Creator values and detect a large number of the targeted documents. More extensive analysis has been done on artifacts associated

with box sizes common in targeted documents. While most of the box sizes as summarized above are not strong indicators themselves, the actual objects they are part of seem to be. It is possible to create more precise signatures by including actual raw values seen in the document, and in some cases, some surrounding data. A handful of these signatures can be constructed to have over 50% TP rate while maintaining a zero FP rate. Manual analysis shows that these signatures, and the artifacts they are derived from, can be found consistently across a large number of documents, which span different embedded malware, different exploits, vastly different structure, and years of time.

Using this knowledge in static signatures is significant because it represents a previously untapped source of signatures that can be used in conventional systems. However, it is conceded that if attackers know this analysis is being performed, that they may learn how to evade these detections, much like malware authors evade other signatures. Even without direct evasion, it may not be feasible to have a large enough data set to vet signatures enough to ensure an acceptably low FP rate.

Where metadata and structural based byte level analysis is likely to have value is in linking related attacks. It is likely that documents that share rare indicators are related either through use of a common tool or common PDF structures used in embedded malware in the document. This view into malware provenance is particularly useful in targeted attacks where understanding of a persistent attacker is sought.

6.3 Quantized Prevalence of Specific Values

When used in conjunction with other features in a machine learner, these specific values show promise to add value when they are consistent. The challenge is to determine if these consistencies can be achieved without hurting detections when metadata values are dynamic or absent.

An important question is how this data would be leveraged in a machine learner. A straightforward implementation would be to use each term as a variable as one would do in SPAM detection or document clustering. The core challenge in doing so is that the count of

these values is very great. It is not clear how one would reduce these terms to an acceptable number of features. Unlike terms in an email body or a document text, there are a very sparse number of each class of metadata/structural item in a given document. It is not feasible to select a subset of terms in this data that will be present in a large number of documents of each class.

One must consider various forms of dimension reduction. For example, one could reduce the Creator field to the core program name while ignoring versions or reduce box sizes to an acceptably low number of box size ranges. Unfortunately, this will not likely be effective, as the specific Creator versions or box sizes are what are useful for correlation. Note that the generic features already include some amount of binned sizes, such as image sizes. One could use some form of statistical learning on a first pass to identify the most important terms, and then use these. This will likely have significant challenges including a still very unwieldy number of terms to select from, lack of stability over time and data sets, and no guarantee that the features selected by one statistical learning method will be optimized for use with the second one actually used for classification.

Another option for using this data would be to only use terms for minority classes, such as targeted documents. This would largely address the dimensionality problem. However, it would limit the value of deviations from the majority classes.

I explored various approaches for converting these values extracted from the documents into usable features that could be used in conjunction with the numeric features already employed. I studied using the feature hashing, but this provided poor performance. I also considered a Bayesian approach, but it provided lower classification quality and higher complexity. I found using binned prevalence based features were more effective than the other methods I tested.

I employed features based on the quantization of the frequency of the term in each class, with the features split by the high level type of the field from which the value is taken. The histogram of the number of documents containing each term is split into buckets, with the bucket which a given term falls into forming the basis for these features. The count of the

existence of each pertinent metadata or structural item is calculated on a per document basis as well as aggregated into a database providing counts of membership of each value in each class.

The basic types of specific values are split into 10 divisions: creator, producer, author, company, box, image, subject, keywords, title, and file representing their respective metadata and structural items. Furthermore, frequency of observance of the values in each class is divided into a small number (2-5) of buckets. A feature set can be derived from the counts of the items in each bucket. The buckets can be constructed with each class considered independently, or the cross product of the prevalence levels for the classes can be used. For example, when classes are taken independently and three levels are used, the features for the creator items would be:

creator_benign_bucket0, creator_benign_bucket1, creator_benign_bucket2,
creator_malicious_bucket0, creator_malicious_bucket1, and creator_malicious_bucket2.

If the classes are used together to generate buckets, then features for two prevalence levels would be:

creator_benign_bucket0_malicious_bucket0, creator_benign_bucket0_malicious_bucket1,
creator_benign_bucket1_malicious_bucket0, creator_benign_bucket1_malicious_bucket1.

When a new document is to be classified, each value is looked up in the database and is determined to belong in one of the buckets. The count for that bucket is incremented. This is repeated for all present metadata and structural elements.

The primary tunable parameters for this method are the number of levels or quantizations indicating the frequency of a given value, whether the frequency for each class is taken independently or dependently, and the function that used is for partitioning the prevalence parameter.

In general, the function used to define the prevalence based quantization does not matter. This is not surprising as the number of divisions used also had a relatively small impact on the outcome. The following functions were tested: log, sqrt, linear, square, and exp. There

was very little difference between them, but the linear divisions performed the best, so they were used.

Table 6.3 demonstrates the differences in classification error based on if buckets were created independently by class or if the features included the full cross products of the buckets by class and the number of levels or divisions by term prevalence. The error rate is the Random Forests out-of-bag error estimate. Here again, these parameters only have a small affect on classification error rates. Hence, for my evaluations, I employ two prevalence levels and use features based on a single class.

Table 6.3: Classification Error by Feature Set Formulation

Number of Prevalence Levels	Class Dependent (Error %)	Class Independent (Error %)
2	.12	.11
3	.08	.11
4	.07	.12
5	.07	.11

This results of applying this method to the training and operational data sets are shown in table 6.4. I use the Contagio data set for training and the Operational data set to test extrapolation. Using quantized indexes provides mediocre classification rates, which drop as the method is used in extrapolation to less closely related data sets.

Table 6.4: Evaluation of Quantized Indexes

	Cross Validation		Extrapolation	
Method	TPR (%)	FPR (%)	TPR (%)	FPR (%)
Quantized Indexes	99.3	4.2	99.3	13.6

6.3.1 Combining General and Specific Value Methods

It is important to evaluate the effectiveness of specific value methods in conjunction with features based on general attributes. The classification effectiveness of both the quantized and per term forms of specific value indexing are shown in 6.5.

Table 6.5: Combining General and Specific Value Methods

	Cross Validation		Extrapolation	
Method	TPR (%)	FPR (%)	TPR (%)	FPR (%)
Generic Features	99.7	0.20	100	0.28
Specific Value Features	99.3	4.2	99.3	13.6
Generic + Specific Value	99.9	0.20	99.7	0.82

The specific value indexing methods perform especially poorly when extrapolating from one data set to another unrelated data set. Combining generic features and quantized specific value indexes does improve detection rates for cross-validation (TPR rises from 99.7 to 99.9). It also raises the false positive rate (from .28 to .82).

6.3.2 Improving Extrapolation through Prevalence Database Bagging

Thus far, specific value indexing has been shown to be effective for data from similar sources, but has been less effective when operating on more diverse data sets. There is a tendency to high false positive rates when extrapolating to unseen data. This error could be explained by poorer durability in the underlying features, or it could be caused by generation of a machine learner that extrapolates poorly due to overfitting, for example.

To improve the predictive capabilities of specific value indexes, the training process was modified such that the database used to look up terms during feature generation was populated with a subset of documents from the training set. The full training set is used to generate the feature set for the Random Forests classifier, but the database of specific values from which specific values prevalence rates are taken only contains a portion of the training set. Ergo, during training, many of the specific values found in the documents in the training set are not found or have a lower prevalence than they would have otherwise, tending to diminish the classification error related to previously unseen values.

The results of modulating the amount of the training set in the specific value prevalence database is shown Table 6.6.

Note that if only half the training set is represented in the specific value incidence

Table 6.6: Varying Portion of Training Set in Prevalence Database

Subset Size (%)	True Positive Rate (%)	False Positive Rate (%)
100	99.3	14
80	93.6	1.6
66	93.6	0.72
50	93.6	0.55
33	15.5	0.57

database, then the high false positive rate that occurs when extrapolating to unseen data dramatically diminishes. This suggests that these features are durable enough to be useful for classification, but the classifier needs to be trained with tolerance for unseen data. When the data in the prevalence database is bagged, specific value indexing is shown to be effective.

This improvement also causes the combination of the specific value features and the generic features to perform much better than either alone. This improvement is demonstrated in Table 6.7.

Table 6.7: Combining Specific Value and General Features with Bagging of Prevalence Data

	Cross Validation		Extrapolation	
Feature Set	TPR (%)	FPR (%)	TPR (%)	FPR (%)
Generic	99.7	0.20	100	0.28
Specific Value	99.6	0.20	93.6	0.55
Generic + Specific Value	99.8	0.10	99.7	0.16

When optimized, specific value indexing can be effective at reducing a portion of the residual classification error from classification based on general features of PDF documents. It was determined that using a quantized approach to generating features based on specific value prevalence is most effective. Also, using bagging during insertion to the specific value prevalence database allows the classifier to be more robust to typical differences in documents and perform better when extrapolating to previously unseen data. The challenge to this approach is that the majority of these values should be relatively easy to modify as

they are generally coincidental with adding malware to documents.

Chapter 7: Microsoft Office Document Malware Detection

The ability to detect malware using features based on document structure and metadata is not unique to the PDF file format. To demonstrate the universality of this approach, the same methodology is applied to other file formats, particularly the Microsoft Office file formats. I demonstrate that it is possible to classify both OLE and zip based files using features derived from these file containers.

7.1 Detection of OLE Based Malware Using Metadata and Structure

I performed a much less rigorous study using mechanisms similar to PDFrate on the Microsoft Office (OLE) file format. The same general methodology was applied to the Microsoft Office file format, including development of metadata extraction techniques, identification of features (231 were used), and the application of the same statistical learning routines. The features used for OLE files were similar to that used for PDF but were peculiar to the OLE format. For example, the number and sizes of embedded streams, the number and type of embedded image files, and the count of embedded fonts and tables are used as features.

The data set for these preliminary results consisted of 10,000 documents obtained from a combination of the same sources as the Contagio and operational PDF data sets, but is compiled without regard for maintaining separation of the two data sources. This data set is summarized in Table 7.1. The outcome of this testing is shown in Tables 7.2 and 7.3 which show the classification confusion matrix for the internal estimate of error performed during classifier construction for each facet of classification. The overall classification error estimate is 0.45% for ben/mal and 1.47% for opp/tar.

Table 7.1: OLE Data Set Summary

Class	Count
benign (ben)	9,592
opportunistic (opp)	352
targeted (tar)	56
total	10,000

Table 7.2: OLE Classification Matrix (ben/mal)

Class	Count	Count
	ben	mal
ben	9587 (TN)	5 (FP)
mal	40 (FN)	368 (TP)

Table 7.3: OLE Classification Matrix (opp/tar)

Class	Count	Count
	opp	tar
opp	349 (TN)	3 (FP)
tar	3 (FN)	53 (TP)

Similar to PDF documents, it is anticipated that detection rates can be increased through improvements to the feature extraction and data set compilation. Regardless, these basic results demonstrate strong promise in classifying OLE files.

7.2 Detection of Zip Based Malware

There are numerous file formats based on the zip container. I studied the ability to classify various file formats using features from the zip container. Specifically, I studied classification of Office 2007 OOXML (.docx), Java Archives (.jar), and android packages (.apk).

7.2.1 Zip File Attributes

Zip files contain a fair amount of metadata about each file contained inside of them. Generally, they contain information similar to a filesystem such as dates and attributes. Zip files also contain information related to how the files are compressed, including the compression type and the version of the zip specification needed to extract the file.

7.2.2 Zip Features

From the few attributes contained in the zip metadata, 99 features were constructed. The most important features by file type are shown in Table 7.4.

There are a few classes of features that are ranked highly across all zip based file types. The "file type" predictors are based on the prevalence of several categories of file types derived from inspection of the extension of the file name. For example "file type img" indicates the prevalence of various types of images including .gif, .jpg, and .png image files. The "dir depth" features indicate the prevalence of files at the specified level of the directory tree in the archive. For example, "dir depth 0" indicates the prevalence in the root of the zip archive. The "compress ratio" features indicate the prevalence of files whose compressed size to raw size are below the threshold specified. For example, "compress ratio 20" indicates the prevalence of files whose ratio of compressed size to raw size is between 0% and 20%. The features that end with "first" or "last" indicate relative position of the first or last file in the archive that contains the indicated attribute. For example, "core.xml last" indicates if the last file in the archive is named "core.xml". "extended field" features indicate the prevalence of specific extended fields.

The effectiveness of these features can be split into two main heads. Many of the features reflect attributes of the files making up the zip archive. Some of the features are artifacts of the program used to create the zip archive, which can also be useful for classification.

Most of the features that are useful for classification reveal attributes of the files contained in the zip archives. For example, the features based on file type reveal the type of files contained in the archive. In the cases of .docx files, the existence of .bin files is

Table 7.4: Zip File Format Top Features

docx	jar	apk
file type bin	dir depth 0	ext cafe last
compress ratio 20	create version 2.0	type sig last
file type xml	compress method store	extended field 0xCAFE
extended field 0xA220	dir depth 2	AndroidManifest.xml last
dir depth 0	compress method deflateN	compress ratio 100
dir depth 1	file type class	dir depth 2
dir depth 3	num dir entries	classes.dex last
Content Types.xml last	dir depth 1	create version 2.0
compress ratio 60	file type oth	dir depth 1
dir depth 2	extended field 0xCAFE	compress ratio 40
compress ratio 40	compress ratio 40	file type oth
file type oth	ext cafe last	file type xml
core.xml last	file type xml	compress method deflateN
num dir entries	dir depth 4	compress ratio 60
compress ratio 80	file type img	compress ratio 80
compress method deflateS	compress ratio 120	file type img
app.xml last	compress ratio 80	num timestamps uniq
compress ratio 120	file type sig	compress method store
compress method store	num timestamps uniq	dir depth 0
file type img	compress ratio 60	file type dex

important because dynamic content such as ActiveX content is stored in these files. The only features that provide definitive information about the content of the files in the archive is the compression ratio which is a proxy for the entropy or information density of the files. This is painted with a broad stroke, but does provide the capability to separate highly compressible data such as text from data that does not compress as well.

These features provide high classification quality based on a relatively small amount of data and superfluous analysis of the zip archive. The durability against direct evasion of these features is not well known and can probably only fairly be evaluated with extended research. However, there are a few reasons why these features are rooted in attributes that are not completely arbitrary and, therefore, not trivially evaded by an attacker. Typically, the declared file extension is not to be trusted. In many cases the method of file access and conventions of the file format may enforce some consistency. For example, the .jar

Table 7.5: Zip File Classification Result by Type

	.docx			.jar			.apk		
	ben	mal	error	ben	mal	error	ben	mal	error
ben	399	16	3.9%	4745	271	5.4%	1780	184	9.4%
mal	26	389	6.3%	163	4853	3.2%	396	1568	20.2%

format generally requires Java class files to be named as .class files to be loaded through the typical class loading mechanisms. Similarly, the information density of some content may be difficult for an attacker to normalize. For example, it may be difficult or even prohibitive to convert malicious code to match the information density of other content, especially if decoding of the content is not practical because of how the content is used in the exploit. Some malicious file attributes may be relatively easy for an attacker to mimic. For example, the tendency of malicious files to have simpler directory structure could be overcome by an attacker by merely adding superfluous files and directories the malicious archive. If this additional inert content does not make delivery of the exploit more difficult, then this class of features may be easy to evade.

7.2.3 Zip Based File Format Classification Evaluation

The effectiveness of the classification for each file type is shown in Table 7.5. The data for this evaluation is taken from VirusTotal, using equal parts benign and malicious samples. The results are taken from the Random Forests out-of-bag error estimate.

This demonstrates that just operating on zip features, it is possible to provide surprisingly high accuracy in classifying various zip based file formats. For example, for .docx and .jar, the accuracy is near 95% and for .apk it is near 85%.

7.2.4 Zip Creator Fingerprinting

Some features derive their utility from how the zip archive is constructed and, therefore, reflect characteristics of specific archive creation mechanisms or programs. The features may reflect a known malicious zip archive creator or the features may indicate that a known

benign creator or set of benign creators was not utilized. The basis of classification is not then the content in the zip file, but the way the zip file is packaged.

Some zip archive programs create zip files with attributes that allow them to be discriminated. At a high level, this is similar to other forms of software fingerprinting including OS network stacks, web browsers, and web server software. These include the zip version attributes, the compression algorithms used, the dates set, permission attributes, various facets of extended fields, the inclusion or exclusion of specific files, and the order of files in the archive. Despite all these attributes, many of the zip creators use null, default, or the same values for these attributes, resulting in the inability to uniquely fingerprint every zip creator program. However, some generally benign and some generally malicious zip creators can be successfully identified by the zip files they create.

The following demonstrates a basic .docx file that was created with Microsoft Office as reported by the zipinfo utility:

```
-rw---- 4.5 fat    1312 b- defS 80-Jan-01 00:00 [Content_Types].xml
-rw---- 4.5 fat     590 b- defS 80-Jan-01 00:00 _rels/.rels
-rw---- 4.5 fat   17187 b- defS 80-Jan-01 00:00 word/document.xml
-rw---- 4.5 fat    6994 b- defS 80-Jan-01 00:00 word/theme/theme1.xml
-rw---- 4.5 fat    1797 b- defS 80-Jan-01 00:00 word/settings.xml
-rw---- 4.5 fat    1295 b- defS 80-Jan-01 00:00 word/fontTable.xml
-rw---- 4.5 fat     677 b- defS 80-Jan-01 00:00 word/webSettings.xml
-rw---- 4.5 fat     713 b- defS 80-Jan-01 00:00 docProps/app.xml
-rw---- 4.5 fat     641 b- defS 80-Jan-01 00:00 docProps/core.xml
-rw---- 4.5 fat   15148 b- defS 80-Jan-01 00:00 word/styles.xml
```

Attributes that are strongly characteristic of Microsoft Office zip packaging include the zip version of 4.5, the use of deflateS zip algorithm, and the date set to the windows epoch. In contrast, the following excerpt from a malicious .docx file leveraging the cve-2013-3906 shows differences in these attributes:

```
-rw---- 4.5 fat    1296 b- defS 80-Jan-01 00:00 word/fontTable.xml
-rw-a-- 6.3 fat   19098 bx defN 13-Mar-23 13:57 word/media/image2.tiff
-rw---- 4.5 fat    1584 b- defS 80-Jan-01 00:00 word/settings.xml
-rw---- 4.5 fat   15542 b- defS 80-Jan-01 00:00 word/styles.xml
-rw---- 4.5 fat    7043 b- defS 80-Jan-01 00:00 word/theme/theme1.xml
-rw---- 4.5 fat     260 b- defS 80-Jan-01 00:00 word/webSettings.xml
-rw-a-- 6.3 fat    6678 bx defN 13-Apr-06 19:48 [Content_Types].xml
-rw---- 4.5 fat     590 b- defS 80-Jan-01 00:00 _rels/.rels
-rw-rw- 2.0 fat 2097088 b- defN 13-Apr-26 16:28 word/activeX/activeX.bin
-rw-rw- 2.0 fat  482542 b- stor 13-Jul-08 04:37 word/activeX/activeX1.bin
```

Note that the files added/modified to enable the exploit have different zip versions, varying dates, differences in permissions and attributes, and that the order of the files ([Content_Types].xml should be first file) is different than would be the case if this zip was created with Microsoft Office.

Characteristics that are normally tied to benign or malicious zip archive creators must be used cautiously, however. Detections based on these characteristics are false negative prone because not all malicious files necessarily deviate from the norm for benign files. For example, in the case of malicious .docx files, it was observed that all the files that involve an exploit have zip packaging characteristics that do not align with Microsoft Office. However, there were a large number of malicious documents that used macros instead of true exploits. The majority of these macro based maldocs had zip characteristics consistent with Microsoft Office with one notable exception being those created with Metasploit. Detections based on artifacts of the creation program are also false positive prone. While the vast majority of benign documents are created by Microsoft Office, other applications, such as Libre Office and Polaris Office, are commonly used to create benign documents that do not match the zip characteristics of Microsoft Office.

These attributes of zip archive structure that aid classification are superficial. Whether indicative of differences from benign creators or artifacts left by malware generation tools,

countering detections based on these attributes would be achieved by mimicking typical or benign zip archive attributes. This is not particularly difficult. It would be straightforward to create a utility to modify these attributes by re-creating zip archives with typical characteristics.

7.3 Discussion

Detection of malfeasance is possible using attributes of zip archives in various zip based file formats. Classification accuracy can be as high as 95%. However, features of this type have not been shown to be durable: many should be susceptible to mimicry evasion with varying degrees of difficulty on the part of the attacker. As such, these features based on zip attributes could be suitable for inclusion in a classifier considering features from other sources.

Chapter 8: Conclusion

8.1 Summary

In this thesis, I sought to defeat exploits through modifications to commonly exploited file formats. I also studied detection of malicious documents using structural and metadata features and a machine learning based classifier. Responding to evasion attacks against PDFrate, I used diversity in ensemble classifiers to identify possible evasion.

I found that document content randomization, where content storage is modified at the file format level without changing the logical representation, is effective in blocking many exploits in Office documents. It is possible to randomize the block layout in OLE files and the encoding method in zip files, mangling exploit data in both document files and reader memory. This method is employed between document creation and document opening such that no modifications to the reader program or operating system are required. The overhead is comparable to signature matching. Document content randomization is applicable to situations where exploits use improper access to document content.

Recognizing that not all exploits can be prevented, I sought to improve malware detection rates using a Random Forest based classifier and structural features from documents. PDFrate provides high classification accuracy including detection of novel malware samples that evade signature based detectors and separation of targeted attacks from broad based malware.

Due to the accuracy and availability of PDFrate, it has been the target of numerous recently published evasion studies. The attacks in these studies employed addition of decoy elements (mimicry), minimization of malicious content (reverse mimicry), and feature extractor subversion. Most attacks against PDFrate cause the classifier voting score distribution to fall between that of benign and malicious documents, making them easy to

differentiate as outliers. I introduce mutual agreement analysis, where the level of consensus in the votes from individual trees in the Random Forest is measured. Mutual agreement in ensemble classifiers serves as an effective estimate of classifier confidence for each prediction. Mutual agreement analysis applies generally to situations in which classifiers are subject to mimicry attacks and helps optimize retraining.

8.2 Lessons Learned

In this thesis, I learned the following:

Content Randomization Based Exploit Protections Modifications to input data, such as randomization of content in Office documents, can foil exploits by inducing entropy in raw file access and reader memory. It is possible to permute data in common file formats without changing the representation presented to the end user.

Structural Feature Based Malware Detection Machine learning based classifiers can provide high malware detection rates for files such as PDF documents. A robust feature set can be extracted from PDF document structure and metadata. Using a Random Forest classifier, reliable detection of previously unseen malicious documents can be achieved.

Mutual Agreement Analysis Identifies Classifier Failures Measuring the level of concurrence in the individual votes in an ensemble provides a measure of confidence of predictions. Feature bagging is critical to creating the entropy in an ensemble that provides evasion resistance. Using mutual agreement analysis, most classifier failures due to novel malware or mimicry attacks can be identified.

8.3 Limitations

I recognize the following limitations in this work:

Content Randomization Applicability Document Content Randomization is applicable when exploits rely on misuse of document content. The use of document content in malware is often driven by exploit protections such as ASLR, prohibitions on scripting or macros, and obstacles to using externally sourced malware. Office documents fit these parameters, hence content randomization is found to be effective. I did a cursory study of PDF documents and found that some have embedded malware that should be defeated with file level encoding randomization. In many malicious PDFs, javascript is used for heap sprays and the final malicious executable is downloaded from an external source.

Exploit Based Attacks This thesis focuses on defeating delivery of malware through malicious documents. I advance malicious document countermeasures that focus on preventing exploitation of vulnerabilities in the reader application. I do not seek to detect malware propagation that relies on user exploitation. The mechanisms proposed do not detect macro based malware, which is very common in Office documents, and other social engineering based attempts to convince users to execute malware contained in documents. Furthermore, I do not address documents used to exploit human vulnerabilities, such as those that instruct users to visit malicious websites or transfer money. Hence, this thesis is limited to malware that exploits a software vulnerability, and the mechanisms presented have limited applicability to other forms of malicious content.

Superficial Feature Extractor and Deployment As demonstrated by the Mimicus attacks explained in Section 5.2.1, my feature extractor can be fooled by spoofed document artifacts. Approaches that employ more complete parsing could prevent these forms of evasion. As shown in Section 5.2.3 and Section 5.2.4, limited or incorrect parsing of documents can cause incomplete feature extraction and can enable classifier evasion. Also, both my exploit mitigation techniques and malicious document detectors are stand alone systems that operate on documents. These mechanisms require a system that provides raw files for analysis, such as Laika BOSS [1] which is used

[1] https://github.com/lmco/laikaboss

with PDFrate in practice. The mechanisms presented in this thesis require access to raw document files for operation and rely on effective feature extraction for reliable detection.

Ground Truth My evaluation relies on knowledge of the classification of a large number of documents. It is not trivial to determine ground truth for large numbers of documents. I rely on antivirus engines after a waiting period to determine if documents are benign or malicious. This is very effective for known malware, but it has been shown that antivirus engines can fail to detect targeted malware for years [53]. Determining ground truth for classification as opportunistic malicious or targeted malicious is much more difficult, as the difference between these two attack groups can be difficult to define, let alone determine. Very often, the distinction is made using factors loosely related to the document itself including information in the embedded malware or patterns in the delivery vector or targeting. The accuracy of my evaluation rests in the validity of my ability to obtain ground truth on these samples.

8.4 Future Work

Building upon this thesis, the following are topics for future work:

Input Based Exploit Protections Document Content Randomization prevents some exploits. Other forms of input based exploit protections and additional file formats should be studied. For example, greater investigation of compression based data permutations is warranted. Improving file format specific exploit protections using modifications to the document format or reader program operation is an area of future work. For example, designing file formats that facilitate or prevent use of file content in exploits is a possible research topic.

Structural and Metadata Based Features In this thesis, I demonstrate that the structural and metadata based approach provides high classification accuracy for PDF files. I conduct only a limited evaluation on other file formats. Understanding the degree

to which other file formats are amenable to this technique is a natural extension of this work.

Combining Structural Features with Other Attributes I advance the use of features based on structure and metadata in documents. Future work should explore these features combined with features taken from sources such as document content, dynamic analysis, malware delivery infrastructure, and victim specific targeting. Since feature extractor subversion is major evasion vector, features that indicate extractor failures is an additional feature type that should be explored.

Mutual Agreement Analysis Refinement Mutual agreement analysis is shown to be effective through evaluation on evasion attacks against real malware detectors. The general applicability of mutual agreement analysis could be better demonstrated with a study of a larger number of evasion attacks on machine learners. A more comprehensive comparison with other methods of outlier detection should be performed.

Mutual Agreement Aware Adversarial Learning This thesis demonstrates that mutual agreement analysis is effective at distinguishing contemporary evasion attacks. Mutual agreement analysis has not been subjected to direct adversarial learning. For example, the degree to which mutual agreement analysis enables training set poisoning should be studied. Mutual agreement analysis will likely be employed to improve mimicry attacks. Effective evasion of mutual agreement, and countering this evasion, is an important topic that will likely be addressed in future research.

Appendix A: Examples of Malicious PDF Structure

Malicious PDF documents present a wide diversity in structure. Presented here are two distinct malicious documents with large differences in metadata and structure, despite using the same exploit: CVE-2009-4324. The salient structural elements of these documents, one targeted malicious and the other opportunistic malicious, are represented in Table A.1 and Table A.2 respectively.

The targeted document is large (4MB) and was delivered via a targeted email. It was rated by the classifier as 84.4% malicious and 80% targeted. The targeted document has many structural elements and attributes, including text content and font objects, that are superfluous to successful exploitation. Indeed, even the content in these unnecessary elements, which would not be seen by the user, is inconsistent with the social engineering used in this attack. Concerning metadata, the targeted document contains PDF ID values, which is normal. However, these values are the same which indicates that this document was not modified by a conventional PDF editor, which should change the ID1 value but leave the ID0 static. Upon successful exploitation of the reader program, the shellcode decrypts and drops a malicious windows portable executable and a benign PDF document. The existence of benign document artifacts and the method of embedding the malicious payloads in the targeted document suggest the author constructed the document from an existing benign document or document template. Construction was likely performed without a conforming PDF editor as evidenced by PDFID metadata and invalid streams.

The opportunistic document is very small (2 KB) and was delivered through malicious web traffic. The classifier assigned a rating of 100% malicious, 0% targeted (100% opportunistic). This document is minimal in structure. It has only the few structural elements necessary for obfuscation and exploitation. This document has no valid optional metadata. However, an object labeled as the "Creator" metadata item houses the bulk of the malicious content in the document and comprises 70% of the document. Regardless, the "Creator"

metadata item is not reported by PDF metadata extraction tools. When exploitation occurs, the necessarily small shellcode pulls additional malicious content from the Internet. Contrasting with the embedding seen in the targeted document, the streams containing the exploit code must be decoded by the vulnerable reader, so these streams are well-formed.

Table A.1: Example Structure: Targeted

Location	Content	Description
000000 000020 000040	%PDF-1.5..%.....1 0 obj..<</Pages 2 0 R /Type/Catalog/OpenAction 8 0 R >>..endobj..2 0 obj..<</	PDF header and OpenAction object which executes javascript when document is opened
000140 000160	j..4 0 obj..<</Type/Font/BaseFont/Times-Roman/Subtype/Type1>>..e	Font object: Times Roman.
000180 0001A0 ... 000390	ndobj..5 0 obj.. <</Length 471/Filter/FlateDecode >>stream..x.U..n ... .endstream..endobj..6 0 obj<< (Decoded, extracted raw text): Financial Reform Puts Republicans on the Spot (April 26) -- Even as they lost today's Senate vote	Object containing formatted text. The victim never sees this content. This content is not consistent with rest of social engineering used in attack.

0003E0 000400 000420	.endobj..7 0 obj..<</Type/Font/Subtype/Type1/BaseFont/Helvetica/Encoding/WinAnsiEncoding>>..endo	Font object: Helvetica.
000500 000520 000540 000560 000580	n..trailer..<</Size 9/Prev 01906 97/Root 1 0 R/ID[<5181383ede9472 7bcb32ac27ded71c68><5181383ede94 727bcb32ac27ded71c68>]>>..startx ref..0..%%EOF..8 0 obj..<</Type	Document trailer and PDF metadata (PDF IDs). These are likely artifacts of an existing benign document that was used in creation of this malicious document.
0005A0 0005C0 ... 0007D0 ... 000AC0	/Action /S /JavaScript /JS (..function re(count, what) {...var v ... ...this.media.newPlayer(sgo);}.. ... ..Func8x9();..)..>>..endobj..9 0	Malicious javascript object. This javascript exploits CVE-2009-4324.

000AE0 000B00 ... 000F80	obj<</Filter /FlateDecode/Length 1062>>stream..iPh4Code..... ... .endstream..endobj..10 0 obj< (Decoded, Extracted, and Disassembled shellcode): jmp short 0x12 pop edx dec edx xor ecx,ecx mov cx,0x40f xor byte [edx+ecx],0x8e loop 0xa jmp short 0x17 call 0x100000002	Object containing purported compressed stream. This stream is not compressed and contains shellcode. The excerpt shows a JMP-CALL-POP sequence followed by an XOR decryption loop that decodes more shellcode.

000FA0 000FC0 ... 3FF920	</Filter /FlateDecode/Length 114 688>>stream....l.....D.. ... ..endstream..endobj..xref..9 4.. (File Extracted and Decrypted using XOR key of 0xFC, location 000FCD to 0079CD): Type: PE32 executable for MS Windows (GUI) Intel 80386 32-bit Name: update.exe Compiled: Wed Dec 29 02:37:00 2010 (File Extracted and Decrypted using XOR Key of 0xFC, location 007A15 to 3FF921): Type: PDF document, version 1.5 CreationDate: D:20110125105603+08'00' Producer: PDFlib 7.0.3 (C++/Win32) PdfID0: D19C9464650960655B0FB612FD9702E0 PdfID1: D19C9464650960655B0FB612FD9702E0 (Decoded, extracted raw text): Rovos rail - Pride of Africa	Object containing purported compressed stream. This stream is not valid compressed data. The majority of the stream contains two encrypted payloads: a PE executable and a PDF document. The PE executable is decrypted, installed on system, and executed. This malware provides remote access trojan capabilities. The PDF document is decrypted and opened for user. This PDF's content is consistent with rest of social engineering in the attack. When considered in isolation, this dropped PDF is benign.

3FF9A0	12>>..startxref ..1092..%%EOF..	End of document.

Table A.2: Example Structure: Opportunistic

Location	Content	Description
000000	%PDF-1.0..1 0 obj<</Type/Catalog	Document header.
000120 000140 000160 000180 ... 000690	>>endobj..7 0 obj..<</Creator 8 0 R>>..endobj..8 0 obj..<</Length 1299 /Filter /FlateDecode..>>. .stream..x..Y...(..R.‘...k:.‘..? ... .bw...endstream..endobj. (Deflated and un-escaped to reveal javascript excerpt): function a(){util.printd(’p@1111 1111111111111111111 : yyyy111’, new Date());}var h = app.plugIn ... d + e;}a();a();try {this.media.newPlayer(null);} catch(e) {}a(); (Further un-escaped to reveal shellcode excerpt): t....URLMON.DLL. URLDownloadToFil eA.pdfupd.exe.crash.php.http://111.gosdfsdjas.com/l.php?i=16..	Reference to, definition of object representing the ”Creator” metadata item. This object contains a compressed stream. The compressed stream contains obfuscated javascript that contains obfuscated shellcode. The javascript exploits CVE-2009-4324. The shellcode downloads more malicious content from the Internet.

0006B0 0006D0 ... 000770	.111611 0 obj<</Filter/FlateDecode/Length 142>>..stream..x...J.* ... a........endstream..endobj..trai (De-obfuscated by removing comments to reveal javascript): var b=this.creator;var a=unescape(b);eval(unescape(this.creator.replace(/z/igm,'%'));</Filter/FlateDecode/Length 142>	Compressed javascript object. This obfuscated javascript un-escapes and executes javascript in "Creator" stream above.
000790 0007B0	ler<</Info 7 0 R /Root 1 0 R /Size 11>>	End of document (without %EOF footer).

Appendix B: PDF Feature Descriptions

Name	Description
author_dot	Author: Count of dot characters
author_lc	Author: Count of lower case characters
author_len	Author: Count of characters
author_mismatch	Count of differences in author values
author_num	Author: Count of numeric characters
author_oth	Author: Count of other characters
author_uc	Author: Count of upper case characters
box_nonother_types	Count of page sized (A4, letter, etc) boxes
box_other_only	Boxes are all other sized (binary)
company_mismatch	Count of differences in company values
count_aa	Count of AA object markers
count_aa_obs	Count of obfuscated AA object markers
count_acroform	Count of Acroform object markers
count_acroform_obs	Count of obfuscated Acroform objects
count_box_a4	Count of A4 sized boxes
count_box_legal	Count of legal sized boxes
count_box_letter	Count of US letter sized boxes
count_box_other	Count of other boxes
count_box_overlap	Count of A4-width, letter-height boxes
count_docid	Count of DocumentID objects
count_encrypt	Count of Encrypt object markers
count_encrypt_obs	Count of obfuscated Encrypt object markers
count_endobj	Count of end of object markers
count_endstream	Count of end of stream markers

count_eof	Count of end of file markers
count_filter	Count of all stream filters
count_filter_A85	Count of A85 stream filters
count_filter_AHx	Count of AHx stream filters
count_filter_ascii85	Count of ASCII85Decode filters
count_filter_asciihex	Count of ASCIIHexDecode filters
count_filter_CCF	Count of CCF filters
count_filter_ccittfax	Count of CCITTFaxDecode filters
count_filter_crypt	Count of Crypt filters
count_filter_dct	Count of DCT filters
count_filter_Fl	Count of FL filters
count_filter_flate	Count of FlateDecode Filters
count_filter_jbig2	Count of JBIG2 Filters
count_filter_jpx	Count of JPX filters
count_filter_lzw	Count of LZWDecode filters
count_filter_LZW	Count of LZW filters
count_filter_mult	Instances streams with multiple filters
count_filter_obs	Instances of obfuscated filter names
count_filter_RL	Instances of RL filters
count_filter_runlength	Instances of RunLengthDecode filters
count_font	Count of Font object markers
count_font_obs	Count of obfuscated Font object markers
count_image_large	Count of images between 786433 and 12582912
count_image_med	Count of images between 64001 and 786432
count_image_small	Count of images between 4097 and 64000 pixes
count_image_total	Count of image objects
count_image_xlarge	Count of images over 12582912 pixels

count_image_xsmall	Count of images between 0 and 4096 pixels
count_instid	Count of InstanceID metadata items
count_javascript	Count of JavaScript object markers
count_javascript_obs	Count of obfuscated JavaScript object markers
count_js	Count of JS object markers
count_js_obs	Count of obfuscated JS object markers
count_launch	Count of Launch objects
count_launch_obs	Count of obfuscated Launch objects
count_obj	Count of object markers
count_objstm	Count of object stream markers
count_objstm_obs	Count of obfuscated object stream markers
count_openaction	Count of OpenAction objects
count_openaction_obs	Count of obfuscated OpenAction objects
count_page	Count of page markers
count_page_obs	Count of obfuscated page markers
count_pdfid0	Count of pdfid0 items
count_pdfid1	Count of pdfid1 items
count_ref	Count of object references
count_ref_nz	Count of non-zero revision object references
count_richmedia	Count of RichMedia objects
count_richmedia_obs	Count of obfuscated RichMedia objects
count_startxref	Count of cross reference table markers
count_stream	Count of stream markers
count_stream_diff	Difference of count_stream and count_endstream
count_trailer	Count of trailer markers
count_xref	Count cross reference table markers
createdate_dot	Count of dot characters in creation date

createdate_mismatch	Count of differences in creation timestamp values
createdate_ts	Creation timestamp (seconds–unix epoch)
createdate_tz	Creation timezone (seconds–UTC offset)
createdate_version_ratio	Ratio of creation date to version (days since Jan 1 1993 / version)
creator_dot	Creator: Count of dot characters
creator_lc	Creator: Count of lower case characters
creator_len	Creator: Count of characters
creator_mismatch	Count of differences in creator values
creator_num	Creator: Count of numeric characters
creator_oth	Creator: Count of other characters
creator_uc	Creator: Count of upper case characters
delta_ts	Difference between creation and modification timestamps
delta_tz	Difference between creation and modification timezones
docid_dot	DocumentID: Count of dot characters
docid_lc	DocumentID: Count of lower case characters
docid_len	DocumentID: Count of characters
docid_mismatch	Count of differences in DocumentID values
docid_num	DocumentID: Count of numeric characters
docid_oth	DocumentID: Count of other characters
docid_uc	DocumentID: Count of upper case characters
docinstid_mismatch	Count of differences in DocumentID and InstanceID values
file_dot	File: Count of dot characters
file_lc	File: Count of lower case characters
file_len	File: Count of characters
file_mismatch	Count of differences in File values
file_num	File: Count of numeric characters

file_oth	File: Count of other characters
file_uc	File: Count of upper case characters
image_mismatch	Count of differences in image dimensions
image_totalpx	Sum of image pixels
instid_dot	InstanceID: Count of dot characters
instid_lc	InstanceID: Count of lower case characters
instid_len	InstanceID: Count of characters
instid_mismatch	Count of differences in InstanceID values
instid_num	InstanceID: Count of numeric characters
instid_oth	InstanceID: Count of other characters
instid_uc	InstanceID: Count of upper case characters
keywords_dot	Keywords: Count of dot characters
keywords_lc	Keywords: Count of lower case characters
keywords_len	Keywords: Count of characters
keywords_mismatch	Count of differences in keywords values
keywords_num	Keywords: Count of numeric characters
keywords_oth	Keywords: Count of other characters
keywords_uc	Keywords: Count of upper case characters
len_obj_avg	Average difference between position of obj and next endobj markers
len_obj_max	Maximum difference between position of obj and next endobj markers
len_obj_min	Minimum difference between position of obj and next endobj markers
len_stream_avg	Average difference between position of stream and next endstream markers

len_stream_max	Maximum difference between position of stream and next endstream markers
len_stream_min	Minimum difference between position of stream and next endstream markers
moddate_dot	Count of dot characters in modification date
moddate_mismatch	Count of differences in modification timestamp values
moddate_ts	Modification timestamp (seconds–unix epoch)
moddate_tz	Modification timezone (seconds–UTC offset)
moddate_version_ratio	Ratio of modification date to version (days since Jan 1 1993 / version)
pdfid0_dot	PDFid0: Count of dot characters
pdfid0_lc	PDFid0: Count of lower case characters
pdfid0_len	PDFid0: Count of characters
pdfid0_mismatch	Count of differences in PDFid0 values
pdfid0_num	PDFid0: Count of numeric characters
pdfid0_oth	PDFid0: Count of other characters
pdfid0_uc	PDFid0: Count of upper case characters
pdfid1_dot	PDFid1: Count of dot characters
pdfid1_lc	PDFid1: Count of lower case characters
pdfid1_len	PDFid1: Count of characters
pdfid1_mismatch	Count of differences in PDFid1 values
pdfid1_num	PDFid1: Count of numeric characters
pdfid1_oth	PDFid1: Count of other characters
pdfid1_uc	PDFid1: Count of upper case characters
pdfid_mismatch	pdfid0 different from pdfid1 (binary)
pos_acroform_avg	Average normalized positions of page markers (% of size)
pos_acroform_max	Normalized position of last acroform marker (% of size)

pos_acroform_min	Normalized position of first acroform marker (% of size)
pos_box_avg	Average normalized positions of box markers (% of size)
pos_box_max	Normalized position of last marker (% of size)
pos_box_min	Normalized position of first box marker (% of size)
pos_eof_avg	Average of normalized positions of last EOF marker (% of size)
pos_eof_max	Normalized position of last EOF marker (% of size)
pos_eof_min	Normalized position of first EOF marker (% of size)
pos_image_avg	Average normalized positions of image markers (% of size)
pos_image_max	Normalized position of last image marker (% of size)
pos_image_min	Normalized position of first image marker (% of size)
pos_page_avg	Average normalized positions of page markers (% of size)
pos_page_max	Normalized position of last page marker (% of size)
pos_page_min	Normalized position of first page marker (% of size)
pos_ref_avg	Average of position of object references
pos_ref_max	Position of last object reference
pos_ref_min	Position of first object reference
producer_dot	Producer: Count of dot characters
producer_lc	Producer: Count of lower case characters
producer_len	Producer: Count of characters
producer_mismatch	Count of differences in producer values
producer_num	Producer: Count of numeric characters
producer_oth	Producer: Count of other characters
producer_uc	Producer: Count of upper case characters
ratio_imagepx_size	Ratio of image_totalpx to size
ratio_size_obj	Ratio of count_obj to size
ratio_size_page	Ratio of count_page to size

ratio_size_stream	Ratio count_stream to size
ref_max_id	Highest numerical value of object references
ref_min_id	Lowest numerical value of object references
size	Size of document (bytes)
subject_dot	Subject: Count of dot characters
subject_lc	Subject: Count of lower case characters
subject_len	Subject: Count of characters
subject_mismatch	Count of differences in subject values
subject_num	Subject: Count of numeric characters
subject_oth	Subject: Count of other characters
subject_uc	Subject: Count of upper case characters
title_dot	Title: Count of dot characters
title_lc	Title: Count of lower case characters
title_len	Title: Count of characters
title_mismatch	Count of differences in title values
title_num	Title: Count of numeric characters
title_oth	Title: Count of other characters
title_uc	Title: Count of upper case characters
url_dot	Title: Count of dot characters
url_lc	Title: Count of lower case characters
url_len	Title: Count of characters
url_mismatch	Count of differences in URL values
url_num	URL: Count of numeric characters
url_oth	URL: Count of other characters
url_uc	URL: Count of upper case characters
version	PDF version as extracted from header

Appendix C: Copyright

Much of the material, both text and graphics, in this dissertation was published previous to inclusion in this work. The material appearing in previously published copyrighted works is duplicated here by permission of the respective copyright holders denoted below.

Some of the content in this thesis, especially that in Chapter 3 was published in [101].

http://link.springer.com/chapter/10.1007/978-3-319-26362-5_11

Some of the content in this thesis, especially that in Chapter 4 was published in [100].

http://doi.acm.org/10.1145/2420950.2420987

Some of the content in this thesis, especially that in Chapter 5 was published in [102].

http://dx.doi.org/10.14722/ndss.2016.23078

Bibliography

Bibliography

[1] PDF Most Common File Type in Targeted Attacks - F-Secure Weblog : News from the Lab. http://www.f-secure.com/weblog/archives/00001676.html.

[2] Shadows in the cloud: Investigating cyber espionage 2.0. http://shadows-in-the-cloud.net/, 2010.

[3] Foreign Spies Stealing US Economic Secrets in Cyberspace, Report to Congress on Foreign Economic Collection and Industrial Espionage, 2009-2011. Technical report, Office of the National Counterintelligence Executive, October 2011.

[4] 5 Attackers & Counting: Dissecting The "docx.image" Exploit Kit. http://www.proofpoint.com/threatinsight/posts/dissecting-docx-image-exploit-kit-cve-exploitation.php, December 2013.

[5] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 41–52, New York, NY, USA, 2006. ACM.

[6] Dmitri Alperovitch. Revealed Operation Shady RAT. http://www.mcafee.com/us/resources/white-papers/wp-operation-shady-rat.pdf, 2011.

[7] R. Amin, J. Ryan, and J. van Dorp. Detecting Targeted Malicious Email. *IEEE Security Privacy*, 10(3):64 –71, May-June 2012.

[8] Matthew Arnao, Charles Smutz, Adam Zollman, Andrew Richardson, and Eric Hutchins. Laika BOSS: Scalable File-Centric Malware Analysis and Intrusion Detection System. https://github.com/lmco/laikaboss, July 2015.

[9] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *21th Annual Network and Distributed System Security Symposium (NDSS)*, February 2014.

[10] M. Bailey, E. Cooke, F. Jahanian, and D. Watson. The Blaster Worm: Then and Now. *IEEE Security Privacy*, 3(4):26–31, July 2005.

[11] Daniel Barbara, Carlotta Domeniconi, and James P. Rogers. Detecting outliers using transduction and statistical testing. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 55–64, New York, NY, USA, 2006. ACM.

[12] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can Machine Learning Be Secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, pages 16–25, New York, NY, USA, 2006. ACM.

[13] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In *Network and Distributed System Security Symposium (NDSS) 2009*, 2009.

[14] Richard Bejtlich. TaoSecurity: What Is APT and What Does It Want? http://taosecurity.blogspot.com/2010/01/what-is-apt-and-what-does-it-want.html, January 2010.

[15] R Beverly and K Sollins. Exploiting transport-level characteristics of spam. In *Conference on Email and Anti-Spam*, 2008.

[16] Sandeep Bhatkar and R. Sekar. Data Space Randomization. In Diego Zamboni, editor, *Detection of Intrusions and Malware, and Vulnerability Assessment*, number 5137 in Lecture Notes in Computer Science, pages 1–22. Springer Berlin Heidelberg, January 2008.

[17] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion Attacks against Machine Learning at Test Time. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, number 8190 in Lecture Notes in Computer Science, pages 387–402. Springer Berlin Heidelberg, 2013.

[18] Battista Biggio, Igino Corona, Blaine Nelson, Benjamin I. P. Rubinstein, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. Security Evaluation of Support Vector Machines in Adversarial Environments. In Yunqian Ma and Guodong Guo, editors, *Support Vector Machines Applications*, pages 105–153. Springer International Publishing, 2014.

[19] Dionysus Blazakis. Interpreter Exploitation. In *WOOT*, 2010.

[20] Stevens Le Blond, Adina Uritesc, Cédric Gilbert, Zheng Leong Chua, Prateek Saxena, and Engin Kirda. A Look at Targeted Attacks Through the Lense of an NGO. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 543–558, San Diego, CA, 2014. USENIX Association.

[21] Stephen Bradshaw. The Grey Corner: Omlette Egghunter Shellcode. http://www.thegreycorner.com/2013/10/omlette-egghunter-shellcode.html, October 2013.

[22] D. Brumley, P. Poosankam, D. Song, and Jiang Zheng. Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 143–157, 2008.

[23] Curtis Carmony, Xunchao Hu, Heng Yin, Abhishek Vasisht, and Mu Zhang. Extract Me If You Can: Abusing PDF Parsers in Malware Detectors. In *21th Annual Network and Distributed System Security Symposium (NDSS)*, February 2016.

[24] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm/, May 2011.

[25] Xiaobo Chen. ASLR Bypass Apocalypse in Recent Zero-Day Exploits. http://www.fireeye.com/blog/technical/cyber-exploits/2013/10/aslr-bypass-apocalypse-in-lately-zero-day-exploits.html, October 2013.

[26] Deepak Chinavle, Pranam Kolari, Tim Oates, and Tim Finin. Ensembles in Adversarial Classification for Spam. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 2015–2018, New York, NY, USA, 2009. ACM.

[27] Corelan Team. Exploit notes-win32 eggs-to-omelet. https://www.corelan.be/index.php/2010/08/22/exploit-notes-win32-eggs-to-omelet/, August 2010.

[28] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and Analysis of Drive-by-download Attacks and Malicious JavaScript Code. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 281–290, New York, NY, USA, 2010. ACM.

[29] G.F. Cretu, A. Stavrou, M.E. Locasto, S.J. Stolfo, and A.D. Keromytis. Casting out Demons: Sanitizing Training Data for Anomaly Sensors. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 81–95, 2008.

[30] Jesse S. Cross and M. Arthur Munson. Deep PDF Parsing to Extract Features for Detecting Embedded Malware. Technical Report SAND2011-7982, Sandia National Laboratories, September 2011.

[31] CrowdStrike. Crowdstrike Intelligence Report: Putter Panda. `http://cdn0.vox-cdn.com/assets/4589853/crowdstrike-intelligence-report-putter-panda.original.pdf`, 2014.

[32] Mark Daniel, Jake Honoroff, and Charlie Miller. Engineering Heap Overflow Exploits with JavaScript. In *Proceedings of the 2Nd Conference on USENIX Workshop on Offensive Technologies*, WOOT'08, pages 1:1–1:6, Berkeley, CA, USA, 2008. USENIX Association.

[33] Theo de Raadt. OpenBSD 3.3 Release Notes. http://www.openbsd.org/33.html, May 2003.

[34] John P. Donaldson. *Source Fingerprinting in Adobe PDF Files.* PhD thesis, Naval Postgraduate Schoo, Monterey, California, USA, December 2013.

[35] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A Survey on Automated Dynamic Malware-analysis Techniques and Tools. *ACM Comput. Surv.*, 44(2):6:1–6:42, March 2008.

[36] Manuel Egele, Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. COMPA: Detecting Compromised Accounts on Social Networks. In *NDSS, 2013*, 2013.

[37] Manuel Egele, Peter Wurzinger, Christopher Kruegel, and Engin Kirda. Defending Browsers against Drive-by Downloads: Mitigating Heap-Spraying Code Injection Attacks. In Ulrich Flegel and Danilo Bruschi, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, number 5587 in Lecture Notes in Computer Science, pages 88–106. Springer Berlin Heidelberg, July 2009.

[38] Jose Miguel Esparza. PDF Attack: A journey from the Exploit Kit to the shellcode, July 2013.

[39] FBI. Update on Sony Investigation. https://www.fbi.gov/news/pressrel/press-releases/update-on-sony-investigation, December 2014.

[40] FBI Internet Crime Complaint Center. Criminals Continue to Defraud and Extort Funds from Victims Using CryptoWall Ransomware Schemes. http://www.ic3.gov/media/2015/150623.aspx, June 2015.

[41] Paul Ferguson. Observations on Emerging Threats. 2012.

[42] Fireeye. APT28: A Window into Russia's Cyber Espionage Operations? https://www.fireeye.com/blog/threat-research/2014/10/apt28-a-window-into-russias-cyber-espionage-operations.html, October 2014.

[43] Prahlad Fogla and Wenke Lee. Evading network anomaly detection systems: formal reasoning and practical techniques. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 59–68, Alexandria, Virginia, USA, 2006. ACM.

[44] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. Bringing Science to Digital Forensics with Standardized Forensic Corpora. *Digit. Investig.*, Digital Investigation, Volume 6:S2–S11, September 2009.

[45] Alexandre Gazet. Comparative analysis of various ransomware virii. *Journal in Computer Virology*, 6(1):77–90, July 2008.

[46] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing Compromise: The Emergence of Exploit-as-a-service. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 821–832, New York, NY, USA, 2012. ACM.

[47] Vinny Gullotto, Joe Faulhaber, Jeffrey Friedberg, Jeff Jones, Jimmy Kuo, John Lambert, Ziv Mador, Mike Reavey, Adam Shostack, George Stathakopoulos, Scott Wu, and Jeff Williams. Microsoft Security Intelligence Report volume 5. Technical report, October 2008.

[48] Mark Handley, Vern Paxson, and Christian Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *2001 USENIX Security Symposium*, pages 115–131, 2001.

[49] Seth Hardy, Masashi Crete-Nishihata, Katharine Kleemola, Adam Senft, Byron Sonne, Greg Wiseman, Phillipa Gill, and Ronald J Deibert. Targeted threat index: Characterizing and quantifying politically-motivated targeted malware. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[50] Cormac Herley. The Plight of the Targeted Attacker in a World of Scale. In *WEIS*, 2010.

[51] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and Mitigation of Peer-to-peer-based Botnets: A Case Study on Storm Worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, pages 9:1–9:9, Berkeley, CA, USA, 2008. USENIX Association.

[52] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 43–58, New York, NY, USA, 2011. ACM.

[53] Mikko Hypponen. Why Antivirus Companies Like Mine Failed to Catch Flame and Stuxnet. http://www.wired.com/2012/06/internet-security-fail/, June 2012.

[54] iSIGHT Partners. NEWSCASTER - An Iranian Threat Inside Social Media. Technical report, May 2014.

[55] S. Jana and V. Shmatikov. Abusing File Processing in Malware Detectors for Fun and Profit. In *2012 IEEE Symposium on Security and Privacy (SP)*, pages 80–94, May 2012.

[56] Jiyong Jang, David Brumley, and Shobha Venkataraman. BitShred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 309–320, New York, NY, USA, 2011. ACM.

[57] Georgios Kakavelakis, Robert Beverly, and Joel Young. Auto-learning of SMTP TCP Transport-Layer Features for Spam and Abusive Message Detection. In *LISA 2011, 25th Large Installation System Administration Conference*, Boston, MA, USA, December 2011. USENIX Association.

[58] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, AISec '15, pages 45–56, New York, NY, USA, 2015. ACM.

[59] M. Kanter and S. Taylor. Attack Mitigation through Diversity. In *MILCOM 2013 - 2013 IEEE Military Communications Conference*, pages 1410–1415, November 2013.

[60] Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering Code-injection Attacks with Instruction-set Randomization. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 272–280, New York, NY, USA, 2003. ACM.

[61] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, Digital Investigation Volume 3, Supplement:91–97, September 2006.

[62] Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. In *Machine Learning*, volume 51, pages 181–207, May 2003.

[63] Pavel Laskov and Richard Lippmann. Machine learning in adversarial environments. In *Machine Learning*, volume 81, pages 115–119, August 2010.

[64] Pavel Laskov and Nedim Srndic. Static detection of malicious JavaScript-bearing PDF documents. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 373–382, New York, NY, USA, 2011. ACM.

[65] F. Li, A. Lai, and D. Ddl. Evidence of Advanced Persistent Threat: A case study of malware for political espionage. In *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*, pages 102 –109, October 2011.

[66] Haifei Li, Stanley Zhu, and Jun Xie. RTF Attack Takes Advantage of Multiple Exploits. http://blogs.mcafee.com/mcafee-labs/rtf-attack-takes-advantage-of-multiple-exploits, April 2014.

[67] Wei-Jen Li, Salvatore Stolfo, Angelos Stavrou, Elli Androulaki, and Angelos D. Keromytis. A Study of Malcode-Bearing Documents. In Bernhard Hämmerli and Robin Sommer, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment 2007*, volume 4579, pages 231–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[68] Zhichun Li, M. Sanghi, Yan Chen, Ming-Yang Kao, and B. Chavez. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In *2006 IEEE Symposium on Security and Privacy*, pages 15 pp.–47, May 2006.

[69] Daiping Liu, Haining Wang, and A. Stavrou. Detecting Malicious Javascript in PDF through Document Instrumentation. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 100–111, June 2014.

[70] Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto. A Structural and Content-Based Approach for a Precise and Robust Detection of Malicious PDF Files. In *Proceedings of the 1st International Conference on Information Systems Security and Privacy*, pages 27–36. ScitePress Digital Library, 2015.

[71] Davide Maiorca, Igino Corona, and Giorgio Giacinto. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious PDF files detection. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and*

communications security, ASIA CCS '13, pages 119–130, New York, NY, USA, 2013. ACM.

[72] Davide Maiorca, Giorgio Giacinto, and Igino Corona. A Pattern Recognition System for Malicious PDF Files Detection. In *Proceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition*, MLDM'12, pages 510–524, Berlin, Heidelberg, 2012. Springer-Verlag.

[73] William R. Marczak, John Scott-Railton, Morgan Marquis-Boire, and Vern Paxson. When Governments Hack Opponents: A Look at Actors and Technology. In *Proceedings of the 23rd USENIX Security Symposium*, pages 511–525, 2014.

[74] Joshua Mason, Sam Small, Fabian Monrose, and Greg MacManus. English Shellcode. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 524–533, New York, NY, USA, 2009. ACM.

[75] Dan Mcwhorter. APT1: Exposing One of China's Cyber Espionage Units, 2013.

[76] Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Improving malware detection by applying multi-inducer ensemble. In *Computational Statistics & Data Analysis*, volume 53, pages 1483–1494, February 2009.

[77] Microsoft. Compound File Binary File Format. https://msdn.microsoft.com/en-us/library/dd942138.aspx.

[78] David Moore, Colleen Shannon, and k claffy. Code-Red: A Case Study on the Spread and Victims of an Internet Worm. In *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, pages 273–284, New York, NY, USA, 2002. ACM.

[79] Tyler Moore, Richard Clayton, and Ross Anderson. The Economics of Online Crime. *The Journal of Economic Perspectives*, 23(3):3–20, August 2009.

[80] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats 2008*, pages 7:1–7:9, Berkeley, CA, USA, 2008. USENIX Association.

[81] J. Neyman. Outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 236(767):333–380, 1937.

[82] Nir Nissim, Aviad Cohen, Robert Moskovitch, Asaf Shabtai, Matan Edri, Oren BarAd, and Yuval Elovici. Keeping pace with the creation of new malicious PDF files using an active-learning based detection framework. *Security Informatics*, 5(1), December 2016.

[83] OpenOffice.org. The Microsoft Compound Document File Format. http://www.openoffice.org/sc/compdocfileformat.pdf.

[84] V. Pappas, M. Polychronakis, and AD. Keromytis. Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization. In *2012 IEEE Symposium on Security and Privacy (SP)*, pages 601–615, May 2012.

[85] Mila Parkour. 11,355+ Malicious documents - archive for signature testing and research. http://contagiodump.blogspot.com/2010/08/malicious-documents-archive-for.html, April 2011.

[86] PaX Team. PaX address space layout randomization. http://pax.grsecurity.net/docs/aslr.txt, 2003.

[87] Niels Provos, Moheeb Abu Rajab, and Panayiotis Mavrommatis. Cybercrime 2.0: When the Cloud Turns Dark. *Commun. ACM*, 52(4):42–47, April 2009.

[88] Moheeb Abu Rajab, Lucas Ballard, Noé Lutz, Panayiotis Mavrommatis, and Niels Provos. CAMP: Content-Agnostic Malware Protection. In *NDSS 2013*, 2013.

[89] M. Ramilli and M. Bishop. Multi-stage delivery of malware. In *2010 5th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 91–97, October 2010.

[90] Sami Rautiainen. A look at Portable Document Format vulnerabilities. *Inf. Secur. Tech. Rep.*, 14(1):30–33, February 2009.

[91] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX conference on System administration*, pages 229–238, Seattle, Washington, 1999. USENIX Association.

[92] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *AAAI 98 Workshop on Text Categorization*, July 1998.

[93] Johannes Schlumberger, Christopher Kruegel, and Giovanni Vigna. Jarhead analysis and detection of malicious Java applets. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 249–257, New York, NY, USA, 2012. ACM.

[94] John Scott-Railton, Morgan Marquis-Boire, Claudio Guarnieri, and Marion Marschalek. Packrat: Seven Years of a South American Threat Actor. Technical report, Citizen Lab, University of Toronto, December 2015.

[95] Fermin J Serna. The Info Leak Era on Software Exploitation. *Black Hat USA*, 2012.

[96] Hovav Shacham. The Geometry of Innocent Flesh on the Bone: Return-into-libc Without Function Calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 552–561, New York, NY, USA, 2007. ACM.

[97] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the Effectiveness of Address-space Randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 298–307, New York, NY, USA, 2004. ACM.

[98] Glenn Shafer and Vladimir Vovk. A Tutorial on Conformal Prediction. *Journal of Machine Learning Research*, 9:371–421, June 2008.

[99] C. Shannon and D. Moore. The spread of the Witty worm. *IEEE Security Privacy*, 2(4):46–50, July 2004.

[100] Charles Smutz and Angelos Stavrou. Malicious PDF Detection Using Metadata and Structural Features. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 239–248, New York, NY, USA, 2012. ACM.

[101] Charles Smutz and Angelos Stavrou. Preventing Exploits in Microsoft Office Documents Through Content Randomization. In Herbert Bos, Fabian Monrose, and Gregory Blanc, editors, *Research in Attacks, Intrusions, and Defenses*, number 9404 in Lecture Notes in Computer Science, pages 225–246. Springer International Publishing, 2015.

[102] Charles Smutz and Angelos Stavrou. When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors. In *21th Annual Network and Distributed System Security Symposium (NDSS)*, February 2016.

[103] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 262–271, New York, NY, USA, 2003. ACM.

[104] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305 –316, May 2010.

[105] Yingbo Song, Michael E. Locasto, Angelos Stavrou, Angelos D. Keromytis, and Salvatore J. Stolfo. On the Infeasibility of Modeling Polymorphic Shellcode. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 541–551, New York, NY, USA, 2007. ACM.

[106] Nedim Srndic and Pavel Laskov. Detection of malicious pdf files based on hierarchical document structure. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium 2013*, 2013.

[107] Nedim Srndic and Pavel Laskov. Practical Evasion of a Learning-Based Classifier: A Case Study. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 197–211, Washington, DC, USA, 2014. IEEE Computer Society.

[108] Salvatore J Stolfo, Ke Wang, and Wei-Jen Li. Fileprint analysis for malware detection. *ACM CCS WORM*, 2005.

[109] B. Stone-Gross, M. Cova, C. Kruegel, and Giovanni Vigna. Peering through the iframe. In *2011 Proceedings IEEE INFOCOM*, pages 411–415, April 2011.

[110] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 133–144, New York, NY, USA, 2013. ACM.

[111] L. Szekeres, M. Payer, Tao Wei, and D. Song. SoK: Eternal War in Memory. In *2013 IEEE Symposium on Security and Privacy (SP)*, pages 48–62, May 2013.

[112] S. Momina Tabish, M. Zubair Shafiq, and Muddassar Farooq. Malware detection using statistical analysis of byte-level file content. In *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, CSI-KDD '09, pages 23–31, New York, NY, USA, 2009. ACM.

[113] K. Thomas and D. M. Nicol. The Koobface botnet and the rise of social malware. In *2010 5th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 63–70, October 2010.

[114] Zacharias Tzermias, Giorgos Sykiotakis, Michalis Polychronakis, and Evangelos P. Markatos. Combining static and dynamic analysis for the detection of malicious documents. In *Proceedings of the Fourth European Workshop on System Security*, EUROSEC '11, pages 4:1–4:6, New York, NY, USA, 2011. ACM.

[115] Nart Villenueve, Greg Walton, SecDev Group., and Munk Centre for International Studies. Citizen Lab. Tracking GhostNet: Investigating a Cyber Espionage Network. http://www.infowar-monitor.net/2009/09/tracking-ghostnet-investigating-a-cyber-espionage-network/, 2009.

[116] B. Waske, S. van der Linden, J.A. Benediktsson, A. Rabe, and P. Hostert. Sensitivity of Support Vector Machines to Random Feature Selection in Classification of Hyperspectral Data. In *IEEE Transactions on Geoscience and Remote Sensing*, volume 48, pages 2880–2889, July 2010.

[117] Tao Wei, Tielei Wang, Lei Duan, and Jing Luo. Secure Dynamic Code Generation Against Spraying. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 738–740, New York, NY, USA, 2010. ACM.

[118] R Wojtczuk. The advanced return-into-lib(c) exploits: PaX case study. *Phrack Magazine, Volume 11, Issue 58*, 2001.

[119] Christian Wressnegger, Frank Boldewin, and Konrad Rieck. Deobfuscating Embedded Malware Using Probable-Plaintext Attacks. In Salvatore J. Stolfo, Angelos Stavrou, and Charles V. Wright, editors, *Research in Attacks, Intrusions, and Defenses*, number 8145 in Lecture Notes in Computer Science, pages 164–183. Springer Berlin Heidelberg, January 2013.

[120] Weilin Xu, Yanjun Qi, and David Evans. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In *21th Annual Network and Distributed System Security Symposium (NDSS)*, February 2016.

[121] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging. In *Journal in Computer Virology*, volume 5, pages 283–293, November 2008.

[122] Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, L. Szekeres, S. McCamant, D. Song, and Wei Zou. Practical Control Flow Integrity and Randomization for Binary Executables. In *2013 IEEE Symposium on Security and Privacy (SP)*, pages 559–573, May 2013.

[123] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative {Android} Markets. 2012/February//.

[124] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han. Botnet Research Survey. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 967–972, July 2008.

Curriculum Vitae

Charles Smutz was born in Pendleton, Oregon in 1982. He graduated from Brigham Young University in 2006 with a bachelors degree in Information Technology and minors in Physics and Business. Since 2006, Charles has been employed at Lockheed Martin, focusing primarily on incident response. He began studies at George Mason University in 2006, fulfilling the requirements for a master's degree in Information Security and Assurance in 2009. He completed his Doctorate in 2016.