Adversarial Attacks in Problem Space for VBA Code Samples

Brian Fehrman[1], Francis Akowuah[1], and Shengjie Xu[2]

[1] Department of Electrical Engineering and Computer Science, South Dakota Mines, Rapid City, SD 57701, USA
[2] Department of Management Information Systems, San Diego State University, San Diego, CA 92182, USA

Abstract. Machine learning is being used more frequently in cyber security. One area is in antivirus software. Learning-based antivirus software is not perfect, since threat actors can use adversarial data to craft attacks against these learning-based security products. This paper focuses on using machine learning to craft attacks against Microsoft (MS) Windows Defender. The goal of this work is to provide tooling and knowledge for: security vendors to tighten the security of their products, security professionals to aid in showcasing risks to customers, and law enforcement to help discover the locations and identities of cyber criminals. The attacks focus on modifying Visual Basic for Applications (VBA) code samples to flip Defender's classification of those files. The modifications are done to the code itself (i.e., problem space) rather than just in feature space. The project attempts to extract information from MS Windows Defender in an offline-mode to help craft the attacks. Preliminary results showed that the classification of some of the files was changed after targeted modification and that the attacks were feasible. Additional work will be conducted to further increase the reliability of the attacks.

Keywords: VBA Macros, Adversarial Machine Learning, Oracle Attacks, Evasion Attacks

1 Introduction

Hacking. It's a term most people have heard. When many people think of hackers, they picture a person in a dark room, wearing a hoodie, and typing away on a computer. Hackers weren't always actors with nefarious intentions. In the '50s and '60s, hackers referred to people who toyed with technology to understand how it worked and to help make it work better. It wasn't until the '70s and '80s that the term hacker became synonymous with ill-intent [1]. As time marches forward and technology becomes more pervasive, the presence of hackers that pose a threat to others is only increasing

Each year, Verizon publishes a report that is titled Data Breach Investigations Report (DBIR). The report details statistics on computer security breaches and incidents for the prior year. In the 2023 DBIR, they defined a breach as *"An incident that results in the confirmed disclosure—not just potential exposure—of data to an unauthorized party."* [2] In the same 2023 DBIR report, one point was very clear: one of the prevailing methods that threat actors continue to use to gain an initial foothold in a network is through social-engineering attacks [2]. In a NIST Special Publication [3], the authors defined social-engineering as: *"...a general term for attackers trying to trick people into revealing sensitive information or performing certain actions, such as downloading and executing files that appear to be benign but are actually malicious."* Phishing is a very popular tactic for social engineering campaigns. One form of delivering malware through phishing emails is via Microsoft Office files. MS Office files have a feature known as macros. The intended purpose of macros is to allow users to automate frequent and/or repetitive tasks [4]. A macro, as defined by [4], is: *"a series of commands and instructions that you group together as a single command to accomplish a task automatically"*

The macros are written using a language known as Visual Basic for Applications (VBA). Virtually anything that can be done with a keyboard or mouse can be automated using VBA macros. VBA can perform various tasks, including prompting users for input, modifying the file itself, modifying files on disk, interacting with other programs on the system, and other tasks [5]. Threat actors, however, have found that they can abuse the macro feature of MS Office files to deliver malware. These files are typically delivered inside zip files or as email attachments. These files will often appear to be legal documents, invoices, or other documents that grab the attention of recipients [6]. The macros within these files are capable of deploying the styles of malware listed earlier in this chapter. Researchers from Atlas VPN stated that at one point in 2021, malicious MS Office files accounted for 43% of all malware downloaded [7].

The work described here was originally inspired by a work experience of the first author. In addition to security consulting work for private and government organizations, the first author's employer also works with various law-enforcement agencies to track down and catch cyber criminals. One of the projects that the first author's employer works on is known as *trackback*. The *trackback* project focuses on revealing the true, physical location of cyber criminals regardless of what location-obfuscation technologies they might be using. One of the main techniques for revealing a criminal's location is to send the criminal a weaponized file via social engineering. The criminal is made to believe that the file has another purpose (e.g., providing payment information). When the criminal opens the file, it performs a series of automated tasks to gather information from the criminal's computer. The program then sends the collected information back to the person who is trying to find the criminal.

One family of file types that is used for tracking criminals in the *trackback* project is MS Office files. These MS Office files are weaponized using VBA macros. The VBA macros perform the aforementioned tasks in collecting and sending information that can be used to track the criminal. Although this task is well-intentioned, anti-virus programs cannot differentiate these weaponized

MS Office files from ill-intentioned ones. The first author was brought on to the project to help the MS Office files bypass anti-virus software. In particular, the first author needed to bypass Microsoft Defender Antivirus. For brevity, MS Office files will be referred to as *documents* in the remainder of this paper.

The main goal of this project is to provide knowledge and tooling to help security vendors tighten gaps in their products while also giving law-enforcement entities additional help in finding cyber criminals. The project here will focus on assessing the security of the machine learning models used by Microsoft Windows Defender when inspecting VBA code samples. Machine learning will be used to guide the modification of VBA code samples to change Microsoft Windows Defender's classification of those code samples. To our knowledge, no other research has been published on this exact topic. Microsoft Windows Defender is the focus since it is readily and freely available on modern Microsoft Windows systems. The hope is that the general approach and knowledge could eventually be extended to other antivirus products.

At this point, it would be prudent to discuss our definitions of *feature space* and *problem space*. Feature space in this context refers to the values that result from performing feature extraction against VBA code samples. Problem space refers to the VBA code itself from which the feature values were extracted. Since Defender is the classifier for this work, it is not possible to simply modify the feature values in the feature space. Instead, changing the feature values requires working within the problem space (i.e., changing the VBA code itself).

The remainder of this paper is structured as follows: Section 2 provides background information on Windows Defender, machine learning, and adversarial attacks on machine learning. Section 3 first describes the process of obtaining, cleaning, and then using Defender to classify VBA samples. The rest of Section 3 details the techniques used for modifying the VBA samples in an attempt to change their classification. Section 4 discusses the results of modifying the VBA samples and having Defender reclassify the modified files. Finally, Section 5 overviews the conclusions of the work and provides a road map for future work.

2 Background

2.1 Microsoft Windows Defender

Microsoft Defender Antivirus, previously known as Microsoft AntiSpyware and Windows Defender, has come pre-installed on Microsoft Windows products since Windows Vista. On Windows Vista and Windows 7, it only provided anti-spyware capabilities. Starting with Windows 8, it began providing antivirus services [8]. For brevity, the remainder of this paper will refer to the Microsoft Defender Antivirus as Defender.

Many traditional forms of antivirus programs use signature-based methods to detect malware. This means that the software has seen the malicious file before, recorded a signature of the file using a hashing method, and checks to see if the new file's signature matches the signature of any previously-seen files. If the signatures match, the file is considered malicious [9]. Defender uses this technology as a first line of defense. If the file doesn't match a known-signature, it further inspects the attributes of the file locally to determine if it's malicious. If Defender deems the file as suspicious, it uploads it to Microsoft's cloud architecture where it performs additional static-checks on the file. If the file still can't be deemed safe, it is sent to the final stage in which the file is actually run and inspected within a sandbox environment in the cloud [10].

Microsoft hasn't published the attributes that it inspects within files. According to [10], Defender uses a combination of machine-learning models, generic methods, and heuristic techniques to block previously-seen and new malware. Referring back to the first author's experience in modifying the macro for the criminal-tracking*trackback* project, the first author found that what triggered an alert from Defender appeared to be a moving target. In some cases, having the macro exceed 90 lines of code triggered an alert. The threshold of the number of lines to trigger an alert dropped to just 70 lines when some other attributes of the macro were changed. At this time, we are unsure of what those characteristics were. This moving target provided additional confirmation that a machine learning model was being utilized by Defender for triggering alerts. Microsoft has stated on other occasions that their research teams are focusing on using data collected from computers around the world to build their machine-learning models for antivirus protection [11].

2.2 Machine Learning

Machine learning has a wide variety of applications, including finding content and friends on social media, providing product suggestions on retail sites, recognizing objects or faces in images, performing language translations, and many more [12]. One emerging area in which machine learning is being applied is cyber security [13–15]. In an attempt to stay ahead of the latest threats, vendors are utilizing machine learning for email filtering, endpoint protection, network intrusion detection, and other tasks [16–20].

Despite what vendors might try to convince consumers of, machine learning is not a flawless solution. One method for exploiting flaws in machine learning models is known as *Adversarial Attacks* [21–24]. Adversarial Attacks attempt to trick a machine learning model, poison the data used for training, or extract sensitive data from the model. The work here focuses on Adversarial Attacks, which will be expanded upon in the next section.

2.3 Adversarial Attacks

Although machine learning has been around for over six decades, targeted attacks against machine learning models is a relatively new area of research that has gained momentum within the last decade [25]. Machine learning can be thought of as generally having two phases; the training phase and the testing, or inference, phase. Different attacks target different phases [26]. These attacks can be either open-box or closed-box[3]. In open-box attacks, threat actors have access to all of the data that was used for training and/or the aspects of the machine learning model itself. For closed-box attacks, the attacker does not have access to some or all of the information mentioned for open-box attacks.

Finding an exact, agreed-upon definition for Adversarial Attacks is difficult. In most cases though, researchers do agree that adversarial attacks can be classified into one of four categories:

- Data Poisoning Attacks
- Data Extraction Attacks
- Evasion Attacks
- Oracle Attacks

The work here focused on *evasion attacks* and *oracle attacks.*

Evasion Attacks Evasion attacks happen during the testing phase. These attacks attempt to trick a machine learning model into misclassifying a data sample by making small changes to the data sample. These modified data samples are known as *adversarial examples.* Figure 1 shows an example of an image being modified to fool a machine learning model. The model initially, and correctly, classifies the image as a pig. Small perturbations are then added to the image to form a new image that looks, to a human, nearly the same as the original image. The modified image, however, is now incorrectly classified as an airliner rather than a pig. The pig example is a targeted attack in which the threat actor had a specific label in mind that they wanted to be applied to the adversarial example. An untargeted attack is also possible in which the final label is not important; only that an incorrect label is assigned to the modified data sample [26].

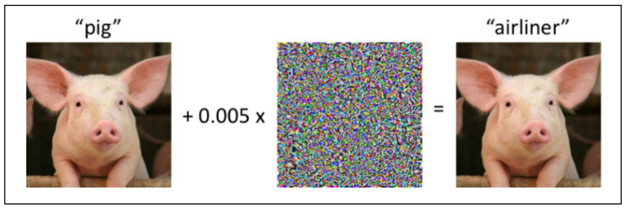

Fig. 1: (Left) Original Image and Classification; (Middle) Changes Added to Original Image; (Right) Modified Image with New Classification. (Reprinted courtesy of the National Institute of Standards and Technology, U.S. Department of Commerce. Not copyrightable in the United States) [26]

In terms of antivirus, samples will likely be classified as benign, suspicious, or malicious. Evasion attacks against antivirus tools that use machine learning

[3] These terms are more commonly known as white/crystal box and black box. In this paper, we chose to replace those terms with the more inclusive and intuitive language of open box and closed box, respectively.

could be either targeted or untargeted. Targeted would mean an attacker wishes that their malware is classified as benign from the start. Untargeted might mean modifying malware so that it is never classified as malicious but may be classified as suspicious at some point in the process. Modification of a binary would mean appending, modifying, or removing instructions or raw bytes from the malware sample. The same technique could be applied to malicious macros in which macros are either appended, modified, or removed.

Oracle Attacks In an oracle attack, a threat actor sends samples to an already-trained, machine-learning model that is now in the testing phase. The threat actor analyzes the responses from the model to attempt to infer information about the model. This information is then used to either reconstruct the model or construct better attacks against the model [26]. Attempting to gain information to reconstruct a model is known, more specifically, as an *oracle extraction attack* [25] or a *model extraction attack* [27]. An oracle extraction attack can also be used to determine which features and feature values a model uses to make certain classifications. The classification boundaries can also be referred to as the model's *decision boundaries* [28].

An oracle extraction attack is, inherently, a closed-box attack. If it were an open-box attack, all of the parameters and structure of the model would be known and there would be no need for the attack. One reward for a successful oracle extraction attack is that the information obtained could be used to craft adversarial examples for malware. If the decision space of a model is inferred, it becomes easier to determine how to modify a malware sample to perform an evasion attack. This attack could potentially be applied as part of a pipeline for evading Defender detection of malicious macros.

3 Attack Path

The attack path described here utilized a combination of evasion and oracle attacks. The steps of the attack path were as follows:

1. obtain a corpus of VBA macros
2. clean the corpus of macros
3. have Defender scan the corpus of macros and perform initial classifications
4. perform feature extraction on the macros based upon previous research and knowledge of VBA macros
5. train a machine learning model on the extracted features
6. determine the most important features and feature values for the model
7. modify the code in the VBA macros to mimic the most values of the most important features
8. repeat steps 3-7 until the classification of the VBA macros changes

3.1 Obtain a Corpus of VBA Macros

Collecting data for machine learning problems can be a problem within itself. The larger the dataset, the more information that can be gleaned and applied to the problem. Large (i.e., approaching 100,000 or more), high-quality datasets are not always readily available to amateur researchers. We have found this to be especially true in the case of VBA macros. VirusTotal[4] is a popular site for obtaining malware samples.VirusTotal, however, can be costly for independent researchers. We were able to obtain temporary academic access. With this access, we obtained a few thousand VBA code samples. One dataset containing approximately 10,000 VBA macros was made available by InQuest in 2021[5].

Through reading other papers, we were made aware of a site named VirusShare[6]. The site is free to use but it does not provide open-registration. Instead, prospective users must contact the site administrator to request access. The first author reached out to the site administrator, who then granted access. The site has a very large collection of malware. One drawback is that the site does not allow for easily searching for and downloading malware of a specific type (e.g., all .docx, .xlsx, etc.) in bulk. What the site does provide is a large list of torrents that point to compressed archives of malware. The contents of the archives are not listed anywhere. The files within the archives also do not have extensions. The archives are also quite large, with some breaking 80GB compressed. Given all of the conditions stated here, an efficient way needs to be determined to download, classify, and keep only the desired document files. A list of all of the torrent links is first obtained from the site. A series of Python and Bash programs were written to perform the following actions:

1. Use *transmission-cli* to download the next archive from the torrent list
2. Decompress the archive using *7z* command-line program and then delete the compressed file
3. Use the *file* utility to determine the type of each file and then *grep* to look for the target types of files
4. Save the desired files
5. Delete the remaining files
6. Repeat the process for the next torrent

The torrent-downloading pipeline was split between four different Digital Ocean nodes. The load was split between systems to speed up the process. Thankfully, the torrents had good download rates at approximately 30 Mb/s per torrent. The majority of the runtime was spent on extracting and classifying the files. Over the course of approximately one week, the distributed pipelines downloaded over 400 archives and obtained over 100,000 potentially-malicious files.

VirusShare appears to frequently upload new archives of malicious files. In between the time of collecting the potentially-malicious files and writing this

[4] https://www.virustotal.com/gui/home/upload
[5] https://github.com/InQuest/microsoft-office-macro-clustering
[6] https://virusshare.com/

paper, VirusShare had added at least ten more archives. The author noted that newer archive files appeared to contain a higher percentage of documents. This is promising in terms of growing the dataset to be used for this work. The author will continue to download new archives as they become available.

The next task in data collection was to obtain potentially-benign documents since malicious files are only one piece of the puzzle. From the existing literature, we discovered that Common Crawl can potentially be used for obtaining a set of benign documents. Common Crawl is a site that continuously crawls and archives the internet[7]. The archived data is made freely available to the public via an API. A GitHub project exists that simplifies the process of searching for specific types of files that are archived by Common Crawl[8]. The author was able to leverage this project to obtain approximately 40,000 potentially-benign files.

3.2 Data Cleaning

Collecting data is only the first step. Cleaning the data is required to obtain meaningful results. The first step of data cleaning was to determine which of the documents actually contained VBA macros since those were the focus of this work. The author wrote a Python program to help with this task. Interfacing with VBA macros within documents in a speedy manner requires the use of proper libraries. The author found that the *oletools*[9] project was open-source and provided both speed and versatility as it was able to handle multiple types of Microsoft Office files.

Upon finding valid VBA code, the same Python program extracted the code and saved it in text format. The first reason for this is that Defender looks at file hashes as part of its pipeline. If files have a hash that matches one that Defender has seen before, it will immediately throw an alert based on the hash. The other reason is that Defender likely looks at other features in a file besides the macros. For instance, the Defender could look at the visible content that is displayed in a document and compare it to content that has been seen in other documents. Since the work focused on the macro aspect, the testing could provide misleading data if Defender were actually alerting on document content rather than the macros themselves. During the macro extraction process, the program also removed duplicate entries.

An additional cleaning program was written in C#. The program took the VBA macros that were extracted by the Python program and performed the following actions:

- removed all comments since it's possible that Defender could be using comments as part of its analysis
- removed all blank lines
- removed files with Unicode so that special processing would not need to be implemented later in the pipeline

[7] https://commoncrawl.org/
[8] https://github.com/centic9/CommonCrawlDocumentDownload
[9] https://github.com/decalage2/oletools/blob/master/oletools/olevba.py

- suppressed files greater than 2KB in size so as to speed up processing and analysis for the initial proof-of-concept for the research
- suppressed code samples less than 100 lines of code so that the code samples would be large enough to be meaningful
- added a small, benign function to the end of each code sample so that the code wouldn't match any current static-signatures that Defender might have

After performing all of the aforementioned cleaning steps, the author's corpus was substantially reduced from over 100,000 files to approximately 15,000 files. The author will continue to add to the corpus as more files are made available on public sources.

3.3 Initial Classification of Samples

The next step was to have Defender perform the initial classification of the VBA macros. Defender does not scan VBA code that is in text-file format. The code must be contained within Microsoft's proprietary, binary format. The author created a blank, macro-enabled document in Microsoft Word. The author then used 7zip to mount the document as an archive to extract out the binary VBA file, *vbaProject.bin*. This file was used as a template into which code samples were inserted. The author used C#, Microsoft's *ole32.dll* library, and the *Kavod*[10] compression library to form binary versions of the VBA text files.

Defender was set up in offline mode for the early stages of this project so more consistent results could be obtained during the testing. A secondary system without public-internet access was used as the scanning system on which Defender was run. The author used C# to create a client and a server program. The client transferred the target files to the remote scanning system. The server program on the scanning system then used the *MpCmdRun* program on Windows to manually run a Defender scan against the target files. The command line below shows the Defender version and the command-line options.

```
C:\> C:\ProgramData\Microsoft\Windows Defender\Platform
   ↪ \4.18.23070.1004-0\MpCmdRun.exe -Scan -ScanType 3
   ↪  -DisableRemdiation -File targetFolder
```

The server program parsed the output of the scanning and returned the results to the client system. The client system then saved the results to a database. The files were either classified as benign or malicious. This produced approximately 11,000 benign files and 4,000 malicious files.

3.4 Feature Extraction

The next step in the pipeline was feature extraction. A public list of good features to use for VBA code analysis was not readily available. The author had to rely largely on their own knowledge of VBA macros and code analysis in general to come up with an initial list of features. Below is a non-comprehensive overview list of the initial features:

[10] https://github.com/rossknudsen/Kavod.Vba.Compression

- number of lines of code
- statistics for the length of each line of code (max, average, min, etc.)
- statistics for function and variable identifier names (length, entropy, upper/lowercase ratio, etc.)
- length of functions, loops, if-statements, and other code blocks
- number of functions, number of variables, and number of references to those items
- use of certain keywords (e.g., *Shell*, *CreateObject*, etc.)

One of the more challenging parts of this project was to parse the VBA code in a way that allowed for extracting the target features. At the time of this project, no libraries for reliably parsing VBA code existed. The closest library for this project was the *Roslyn VB.NET*[11] compiler/parser. The VBA and VB.NET languages share many similarities. This meant that the Roslyn VB.NET could be used in a limited capacity for parsing VBA code. The author wrote a C# program that used the Roslyn VB.NET library to parse the VBA code. The author had to create many workarounds to overcome the limitations of using the VB.NET parser for parsing VBA. The program then extracted the desired features from the parsed code. The program saved the features for each file, along with the file's classification, to a row in a CSV feature file.

3.5 Extracting Feature Importance

The author used Python and the scikit-learn[12] to create an RFC classifier. The author performed randomized undersampling of the benign samples to create a 50/50 split between benign and malicious samples. An 80/20 train/test split was performed on the equalized sample set. This resulted in approximately 6600 samples used for training and 1600 samples used for testing. The RFC classifier achieved an accuracy of 95% with an F1-score of 95% as well.

The Python program then averaged together the feature values for all of the malicious samples in the training set. The program also averaged the feature values for all of the benign samples in the training set. In both the case of both benign and malicious feature value averages, outliers were removed before calculating the averages by using the standard Inter Quartile Range (IQR) outlier removal method. The averaged values for benign samples and malicious samples were saved to their own files. These values formed the target values for later in the pipeline.

A consequence of using an RFC classifier is that it presents the user with a list of the most important features. The author used this list to determine which features to focus on for the evasion attack. Some of the most important features at this stage were shown to be identifier-name length, identifier-name entropy, the ratio of upper to lowercase letters in identifier names, and the number of lines of code.

[11] https://github.com/dotnet/roslyn
[12] https://scikit-learn.org/stable/

3.6 Modifying the VBA Code Samples

The next major challenge of the project was in modifying the VBA code samples. Much of the work on adversarial machine learning has focused on images and video in which retaining code functionality is not a concern [29]. Very little work has been done on modifying source code itself for evasion attacks. The authors in [30] investigated using words from benign VBA files to replace words in malicious files or to insert as arguments to built-in VBA functions (e.g., Ucase, Split, etc.). The work focused on attacking Natural Language Processing (NLP) models that the authors had created. In this case, we were not concerned with retaining functionality of the code as they controlled the process used to analyze the code.

In [31], the authors performed work on creating macros to evade detection by three open-source VBA inspection programs, which were: *mraptor*[13], *olevba*[14], and *ViperMonkey*[15]. The author started their macro from scratch and built it up, rather than modifying existing macros.

There are some tools[16] that exist for performing basic modification of VBA code files, such as obfuscating variables on functions names. These programs did not appear to be well-optimized for dealing with a large number of files.

A program was needed to modify the VBA code samples. A C# program was written and used a combination of the Roslyn VB.NET library and optimized regular-expressions for modifying the code. The program contained functions that inspected the target feature values (determined in Section 3.5) and then attempted to modify the code to match those values. The program is currently able to modify the code in the following ways:

- identifier-name length, entropy, and upper/lowercase ratio
- number of lines of in a function

Modifying Identifiers For modifying identifier names, the program first used Roslyn VB.NET to extract all of the identifier names from the code. The program then iterated through each identifier name and attempted to modify it to match the target values found in Section 3.5. The program used a combination of adding, removing, and modifying characters to achieve the desired values. The program ensured that the identifiers were still valid and that they did not match any of the built-in VBA keywords. The program also ensured that the identifiers were unique. The program performed the modifications in this order: length, upper/lowercase ratio, and then entropy. Entropy was calculated with the Shannon Entropy formula. Although identifiers could be extracted with Roslyn VB.NET, using Roslyn to replace the identifiers proved difficult. The author found that using regular expressions was a more reliable method for replacing the identifiers.

[13] https://github.com/decalage2/oletools/wiki/mraptor
[14] https://github.com/decalage2/oletools/wiki/olevba
[15] https://github.com/decalage2/ViperMonkey
[16] https://github.com/sevagas/macro_pack

Modifying Function Length For modifying the length of functions, the program first extracted all of the functions from the code using Roslyn VB.NET. The program then iterated through each function and attempted to modify it to match the target values found in Section 3.5. The program used a combination of adding and removing lines of code to match the target values.

To add lines of code, the program first defined an identifier of numerical type with a valid name that was randomly generated. The identifier name was then sent through the same length, upper/lowercase ratio, and entropy modification process as the other identifiers to ensure that the features of the identifier were close to the target values. A series of math operations were then performed on the identifier. Each math operation was randomly selected from a list of valid math operations. The math operations were: addition, subtraction, multiplication, and division. Each operation resulted in a new line of code. Operations were added until the function reached the desired length.

To remove lines of code, the program used a function-insertion approach. Lines of code were placed inside of a newly-created function. All of the variables referenced within the relocated code were passed to the function as reference arguments, which meant any changes to the variables in the functions would be reflected if they were used again in the original code. A call to the function was inserted where the lines of code were originally located. If the newly-created functions fell short of the target length for lines of code, the process for adding lines of code was applied to the new functions.

Roslyn VB.NET was used for modifying the functions.

4 Results

The author ran the evasion attack on the set of 3,879 malicious files. The pipeline performed the following modifications on separate runs:

1. Modify the files to match the features of benign files
2. Modify the files to match the features of malicious files
3. Prepend a single character onto each identifier name

Prepending a single character onto each identifier name was conducted as a control to see the difference in classification results when trying to target specific feature values versus simply changing identifier names. In the case of prepending a single character, the length of the functions was not modified. After performing one of the above modifications, the pipeline sent the modified files to the scanning system for classification. The scanning system then sent the results back to the pipeline. The pipeline then saved the results to a database. The pipeline repeated the process for the next modification. Table 1 shows the results of the three runs.

As can be seen in the results in Table 1, targeting feature values rather than just prepending a single character to identifiers had drastically different effects on the classification. The samples in which only a single character was prepended to identifiers were flagged as malicious at a higher rate than the samples in which

Modification	Classified Benign	Classified Malicious
Target Benign Values	659	3220
Target Malicious Values	589	3290
Prepend Z Only	183	3696

Table 1: Results of Modifying Malicious Files

the features were modified to match target values. Targeting benign features values resulted in the least files being classified as malicious. Targeting malicious feature-values resulted in more files being flagged than targeting benign values but fewer than just prepending a single character. There could be a few causes for why targeting malicious feature-values did not produce more flagged files than the character-prepending method. First, it's possible that breaking code up across more functions caused enough confusion for Defender such that it could no longer confidently classify those files as malicious. It's also possible that some of the feature values were close enough between malicious and benign features that attempting to modify some of the files to be malicious actually moves them closer to being benign.

5 Conclusion and Future Work

This paper presents the research motivation, the background of relevant topics, and the current attack path toward performing oracle and evasion attacks on Defender's VBA analysis. This research has obtained a large corpus of VBA macros and has successfully created a pipeline for performing evasion attacks against Defender. The pipeline was able to modify some malicious files to evade detection. The preliminary results are promising for future work in attacks involving creating adversarial VBA code samples that are tested against closed-box classifiers.

Future work will include obtaining or creating additional codes samples to help with ensuring that the feature values that are targeted during code modification are truly representative of either benign or malicious samples. The author will continue improving the modification pipeline to add more modification abilities. Additionally, further investigations will be performed to ensure that the modifications are producing the expected results. The author will release more details and results as the project progresses.

6 Acknowledgement

The authors acknowledge the support from Black Hills Information Security and the South Dakota Governor's Center for Disruption of the Illicit Cyber Economy.

This work was also supported in part by SDBOR CRG award MA2300014. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the South Dakota Board of Regents (SDBOR).

References

1. B. Yagoda, "A Short History of "Hack"," 2014. [Online]. Available: https://www.newyorker.com/tech/annals-of-technology/a-short-history-of-hack
2. Verizon, "Data Breach Investigation Report 2023," *Verizon DBIR*, 2023. [Online]. Available: https://www.verizon.com/business/resources/reports/2023-data-breach-investigations-report-dbir.pdf
3. M. P. Souppaya and K. A. Scarfone, "NIST: 800-114: User's Guide to Telework and Bring Your Own Device (BYOD) Security," *NIST Special Publication*, 2016. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-114/rev-1/final
4. Microsoft, "Create or run a macro," 2021. [Online]. Available: view-source:https://support.microsoft.com/en-us/office/create-or-run-a-macro-c6b99036-905c-49a6-818a-dfb98b7c3c9c
5. ——, "Getting started with VBA in Office," 2022. [Online]. Available: https://docs.microsoft.com/en-us/office/vba/library-reference/concepts/getting-started-with-vba-in-office
6. ——, "Macro Malware," 2022. [Online]. Available: https://docs.microsoft.com/en-us/microsoft-365/security/intelligence/macro-malware?view=o365-worldwide
7. William S. of Atlas VPN, "43% of all malware downloads are malicious office documents," 2021. [Online]. Available: https://atlasvpn.com/blog/43-of-all-malware-downloads-are-malicious-office-documents
8. Computer Hope, "Microsoft Defender Antivirus," 2022. [Online]. Available: https://www.computerhope.com/jargon/w/windefen.htm
9. M. Rezek, "What is the difference between signature-based and behavior-based intrusion detection systems?" 2020. [Online]. Available: https://accedian.com/blog/what-is-the-difference-between-signature-based-and-behavior-based-ids/
10. Microsoft, "Cloud protection and sample submission at Microsoft Defender Antivirus," 2022. [Online]. Available: https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/cloud-protection-microsoft-antivirus-sample-submission
11. A. Ng, "Microsoft is building a smart antivirus using 400 million PCs," 2017. [Online]. Available: https://www.cnet.com/news/privacy/microsoft-build-smart-antivirus-using-400-million-computers-artificial-intelligence/
12. N. Duggal, "Top 10 Machine Learning Applications and Examples in 2022," 2022. [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-applications
13. K. Zhang, S. Xu, and B. Shin, "Towards adaptive zero trust model for secure ai," in *2023 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2023, pp. 1–2.
14. S. Xu, Y. Qian, and R. Q. Hu, "Edge intelligence assisted gateway defense in cyber security," *IEEE Network*, vol. 34, no. 4, pp. 14–19, 2020.
15. S. Chaudhary, A. O'Brien, and S. Xu, "Automated post-breach penetration testing through reinforcement learning," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–2.

16. S. Xu, Y. Qian, and R. Q. Hu, "A semi-supervised learning approach for network anomaly detection in fog computing," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.
17. S. Xu, D. Fang, and H. Sharif, "Efficient network anomaly detection for edge gateway defense in 5g," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–5.
18. S. Xu, Y. Qian, and R. Q. Hu, "Data-driven edge intelligence for robust network anomaly detection," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 3, pp. 1481–1492, 2019.
19. J. Greig, "Eight leading AI/ML cybersecurity companies in 2020," 2020. [Online]. Available: https://www.zdnet.com/article/eight-leading-aiml-cybersecurity-companies-in-2020/
20. S. Xu, Y. Qian, and R. Q. Hu, "Data-driven network intelligence for anomaly detection," *IEEE Network*, vol. 33, no. 3, pp. 88–95, 2019.
21. J. Burr and S. Xu, "Improving adversarial attacks against executable raw byte classifiers," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–2.
22. S. Xu, Y. Qian, and R. Q. Hu, *Cybersecurity in Intelligent Networking Systems*. John Wiley & Sons, 2022.
23. M. Bradley and S. Xu, "A metric for machine learning vulnerability to adversarial examples," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–2.
24. S. Xu, Y. Qian, and R. Q. Hu, "Adversarial examples: Challenges and solutions," 2023.
25. O. Ibitoye, R. A. Khamis, A. Matrawy, and M. O. Shafiq, "The threat of adversarial attacks on machine learning in network security - A survey," *CoRR*, vol. abs/1911.02621, 2019. [Online]. Available: http://arxiv.org/abs/1911.02621
26. E. Tabassi, K. J. Burns, M. Hadjimichael, A. Molina-Markham, and J. Sexton, "A taxonomy and terminology of adversarial machine learning," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:207795576
27. L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3469–3478, 2021.
28. S. Sahu, "Decision Boundary for Classifiers: an Introduction," 2021. [Online]. Available: https://medium.com/analytics-vidhya/decision-boundary-for-classifiers-an-introduction-cc67c6d3da0e
29. I. G. and& Jonathon Shlens and& Christian Szegedy, "Explaining and Harnessing Adverserial ML," *International Conference on Learning Representations (ICLR)*, pp. 1–11, 2015.
30. M. Mimura and R. Yamamoto, "A Feasibility Study on Evasion Attacks Against Nlp-Based Malware Detection Algorithms," 2022. [Online]. Available: https://ssrn.com/abstract=4061102
31. G. Pippi, "Advanced VBA macros: bypassing olevba static analyses with 0 hits," 2020. [Online]. Available: https://www.certego.net/en/news/advanced-vba-macros/