

Article

Not peer-reviewed version

Malicious Office Macros Detection: Combined Features with Obfuscation and Suspicious Keywords

[Xiang Chen](#) , Wenbo Wang , [Weitao Han](#) *

Posted Date: 22 September 2023

doi: 10.20944/preprints202309.1531.v1

Keywords: malicious document; VBA; macro; obfuscation; suspicious keywords; machine learning



Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Article

Malicious Office Macros Detection: Combined Features with Obfuscation and Suspicious Keywords

Xiang Chen, Wenbo Wang and Weitao Han *

PLA Strategic Support Force Information Engineering University

* Correspondence: weitaohanchn@163.com

Abstract: Microsoft has implemented several measures to defend against macro viruses, including the use of Anti-malware Scan Interface (AMSI) and automatic macro blocking. Nevertheless, evidence shows that threat actors have found ways to bypass these mechanisms. As a result, phishing emails continue to apply malicious macros as their essential attack vector. In this paper, we analyze 77 obfuscation features from the attacker's point of view and extract 46 suspicious keywords in macro. We first combine the above two types of features to train machine learning models on a public dataset. Then, we carry out the same experiment on self-constructed dataset, a collection of newly discovered samples, to see if our proposed method could discover the unseen malicious macros. Experimental results demonstrate that, comparing with the existing researches, our proposed method has a higher detection rate and better consistency. Furthermore, ensemble multi-classifiers with distinct feature selection further improve the detection performance.

Keywords: malicious document; VBA; macro; obfuscation; suspicious keywords; machine learning

1. Introduction

Dealing with office documents has become an integral part of work in the digital era. Unfortunately, malicious campaigns utilizing office documents have become commonplace. Typically, an attacker crafts a malicious document that exploits a vulnerability or contains a macro, adds the document as an email attachment, and then sends the email to the target. Since office documents are often perceived as non-threatening, there is a high probability that the malicious document will be opened by the target. However, continuous improvements in platform and application security have resulted in a decline in exploiting vulnerability on software side, attacks based on social engineering and macro abusing have surged in the past decade [1,2]. To counteract this threat, in 2018, Microsoft introduced the Antimalware Scan Interface (AMSI) to enable antivirus and other security solutions to scan macros and other scripts at runtime. It detects malicious behaviour and has integrated into the new version of the Office 365 client application. Last year, Microsoft implemented a proactive measure by blocking macros from the internet in Office[3], which reduces macro-based attack by 66% over the course of eight months[4]. However, as attackers refine their techniques to bypass this mechanism, macro-based Office documents still pose a security threat. Security researchers have observed that attackers are increasingly utilizing alternative document formats, including Publisher files and OpenDocument text (.odt) files. These formats may still execute macros, yet they escape from the latest security controls [6,7]. Despite this, several Advanced Persistent Threat (APT) groups, such as Kimsuky, Donot, and SideCopy, as well as cyber criminals, such as Emotet, continue to rely on macro-based Office documents to launch attacks and distribute their malicious payloads [5].

Current researches focus on detecting malicious macros by examining macro obfuscation and suspicious keywords such as specified strings and functions. Both methods have limitations: Obfuscation detection is unable to distinguish the obfuscation of benign or malicious macros, leading to numerous false positives in detection [8]. The VBA macro runtime is rich in functionality, and new attack techniques are occasionally discovered by attackers. Consequently, this in turn poses a challenge to current detection models, which rely on detecting known suspicious keywords.

In fact, the above mentioned approaches seem to complement each other: Since obfuscation is often necessary for malicious macros regardless of the attack techniques used, obfuscation detection can be used to detect unseen malicious macros, while detection by suspicious keywords can capture known malicious features of macros. However, the above two types of features have never been combined to identify malicious macros in previous researches.

In this paper, we conduct an objective analysis by extracting features of obfuscation and suspicious keywords, and integrate them to detect malicious macros. We thoroughly acquire 123 features extracted from obfuscation and suspicious keywords. Furthermore, we train four widely used machine learning models, each of which selects specific type of features or both. Our method has been evaluated on two datasets: Dataset1 is a public dataset containing 2939 benign samples and 13734 malicious samples. Dataset2 comprises 2885 new samples from Virustotal, all of which were reported after the publication date of Dataset1. The experiments conducted on Dataset1 reveal that adjusted model with combined features achieves the highest F1-score of 0.997. Employing this classifier to detect malicious samples from Dataset2 achieves a detection rate of 95.3%. Comparing with detecting by single class of features, the classifier with combined features enjoys a better performance. Further analysis has shown that we could conduct ensemble training to further improve the detection performance.

This paper contributes in the following ways:

- We extract a concise and consistent set of macro obfuscation features from the perspective of adversarial attack.
- We analyze the drawbacks of using obfuscated features or suspicious keywords in detecting malicious macros, and for the first time combine these two types of features in machine learning models to detect malicious macros. Our approach has a very low rate of false positives and is capable of detecting unseen malicious macros.

The paper is organized as follows. Section 2 outlines the background and details how VBA macros get obfuscated. In Section 3, we present recent researches on detecting malicious macros in Office documents. We propose our approach in Section 4 and evaluate it in Section 5. Finally, we conclude our work and discuss the future work.

2. Background

2.1. Macro in Office documents

Macros, developed in Visual Basic for Applications (VBA)[12] programming language, has been widely applied in Microsoft Office documents such as Word, Excel, PowerPoint, and Outlook. Users develop customized codes interacting with the Office applications, in order to manipulate data, format documents, create charts, send emails, and complete various other tasks. The automation of repetitive tasks stands as one of the principal benefits of utilizing macros. For instance, in Microsoft Word, users need to perform formatting actions several times, including applying a particular font, setting paragraph alignment, or inserting headers and footers. Macros enable users to record these actions only once and conduct whenever necessary, thereby reducing time and effort. They also assist users in accomplishing complex tasks that may otherwise be challenging or time-consuming. For example, Microsoft Excel enables its users to create macros, which analyze large datasets, perform calculations, generate reports, and automatically create charts when triggered by a specific event like workbook opening or button clicking. Figure 1 shows two benign documents' VBA macros. Figure 1(a) demonstrates modifying font format in a word document, while Figure 1(b) shows a macro snippet that automatically sends an email with the document's content.

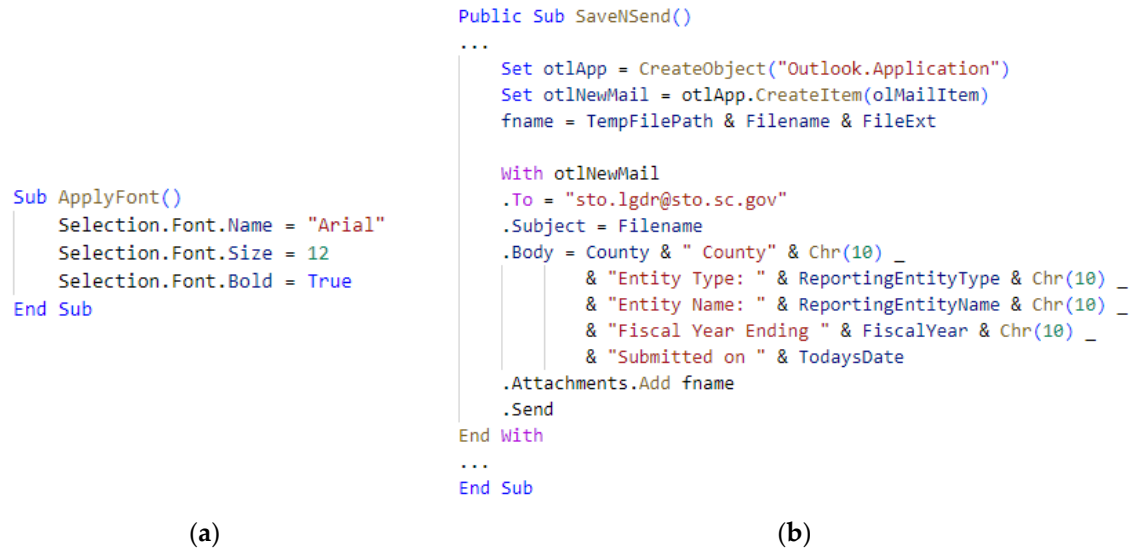


Figure 1 consists of two panels, (a) and (b), showing VBA macro code. Panel (a) shows a sub procedure named `ApplyFont()` that sets the font name to "Arial", font size to 12, and font bold to True. Panel (b) shows a sub procedure named `SaveNSend()` that creates an Outlook application, creates a new mail item, sets the to, subject, and body, adds an attachment, and sends the mail.

```

Sub ApplyFont()
    Selection.Font.Name = "Arial"
    Selection.Font.Size = 12
    Selection.Font.Bold = True
End Sub

Public Sub SaveNSend()
    ...
    Set otlApp = CreateObject("Outlook.Application")
    Set otlNewMail = otlApp.CreateItem(olMailItem)
    fname = TempFilePath & Filename & FileExt

    With otlNewMail
        .To = "sto.lgdr@sto.sc.gov"
        .Subject = Filename
        .Body = County & " County" & Chr(10) _
            & "Entity Type: " & ReportingEntityType & Chr(10) _
            & "Entity Name: " & ReportingEntityName & Chr(10) _
            & "Fiscal Year Ending " & FiscalYear & Chr(10) _
            & "Submitted on " & TodaysDate
        .Attachments.Add fname
    End With
    .Send
    ...
End Sub

```

Figure 1. VBA macros examples. (a) Description of what is contained in the first panel; (b) Snippet of VBA macro used to send a email.

Although bringing convenience for users, macros still help attackers complete different malicious behaviors. As shown in Figure 2, a simple but malicious macro, which consists only four lines, can still accomplish series of malicious actions. It acquires a running instance of Windows Explorer using the VBA function `GetObject`. The method `ShellExecute` is then called to initiate a PowerShell which downloads and executes a malicious script. There are a variety of techniques which can be employed to execute commands, as we would further detail in the following sections.

```

Sub Document_Open()
    Set obj = GetObject("new:C08AFD90-F2A1-11D1-8455-00A0C91F3880")
    obj.Document.Application.ShellExecute "powershell.exe", "IEX (New-Object Net.WebClient).DownloadString('http://192.168.131.127:8000/httpptest1.ps1')", "C:\\Windows\\System32", Null, 0
End Sub

```

Figure 2. Example of malicious VBA macros.

2.2. Obfuscation of VBA Macro

Code obfuscation was proposed as a preventive reverse analysis method. In legitimate situations, macros are typically obfuscated to safeguard the intellectual property of the source code. However, in the context of malicious macros, obfuscation is mostly employed to conceal the malicious commands, making it difficult for antivirus software to identify any suspicious elements and conduct detection. We list six commonly used techniques for obfuscating VBA macros[8].

- **Random Obfuscation:** Randomise the variable names or procedure names in macro with randomly generated characters. Figure 3 provides an example of random obfuscation. Both of the names of variables and function parameters have been obfuscated.

```

Dim COHdDxBvIVN() As Byte: COHdDxBvIVN = oymlTquSfEJJz(gZOnGYtBrHwbfY)
Dim nMLloAyJokfWI As Long: nMLloAyJokfWI = UBound(COHdDxBvIVN)

```

Figure 3. Example of random obfuscation.

- **Split Obfuscation:** Long strings can be split into several short strings connected with "+" or "=". This operation is common in normal macros when a long string needs to be joined by many sub-strings. Figure 4(a) demonstrates split obfuscation in a normal macro, while figure 4(b) shows a snippet of a malicious macro using split obfuscation for the parameter of function `GetObject`.

<pre>If IsError(found) Then msg = msg & rowMsg & " " & cellAddress & " - Invalid, select from list." & vbCrLf End If</pre>	<pre>lw = "" Set obj = GetObject("new:C08A" + lw + "FD90-F" + lw + "2A1-11D1-845" + lw + "5-00A0C91F3880")</pre>
(a)	(b)

Figure 4. Split obfuscation examples. (a) Example of split obfuscation from a benign macro; (b) Example of split obfuscation from a malicious macro.

- **Encoding Obfuscation:** Encoding obfuscation is similar to encryption and requires decoding operations to restore the original string before use. For instance, we can encode a PowerShell command as a byte array or Base64 string. PowerShell commands are frequently targeted for encoding obfuscation. Figure 5 demonstrates the parameter of function `GetObject` is obfuscated using byte arrays and subsequently decoded with a self-defined decoding function, `FicTuJxzQrLu`.

```
Set GswGsBPALDzb = GetObject(FicTuJxzQrLu(Array(((16 Xor 1)+84),(3+110),(6 Xor 11),
(1 Xor 17),112,72),(0 Xor 0)) & FicTuJxzQrLu(Array((13+(50 Xor 172)),139,((14 Xor
21)+51),(29+(3 Xor 15)),(101+98),(223+(19 Xor 5)),(35+(18 Xor 55)),((2 Xor 11)+(13
Xor 20)),((24 Xor 75)+64),((21 Xor 61)+73),148,((36 Xor 176)+(0 Xor 29)),89,(14
Xor 197),(23 Xor 222),((18 Xor 14)+126),(74 Xor 170),((7 Xor 15)+(0 Xor 2)),(7
Xor 23),80,(6+0),(3 Xor 12),(0 Xor 145),((14 Xor 158)+(0 Xor 13)),((35 Xor 20)
+160),(106 Xor 220),(44+(20 Xor 49)),(66+90),92,(87+(58 Xor 184)),173,(15 Xor
111),157,(63 Xor 113)),(5+(0 Xor 1))))
```

Figure 5. Example of encoding obfuscation.

- **Embedding Obfuscation:** Embedding obfuscation conceals the target string in other parts of the macro, such as document attributes, forms or controls, and even within the content of Word or Excel documents. The macro can then directly load this content into a string variable. This obfuscation can be challenging to recognize, as it is a common operation in non-obfuscated macros to process content. Figure 6 provides an example of embedding obfuscation. This PowerShell command is loaded from a hidden shape within the document. The shape is made invisible by some simple settings.

```
'Get the string in the document
content = Mid(ActiveDocument.Shapes.Item(1).AlternativeText, 6)
```

Figure 6. Example of embedding obfuscation.

- **Call obfuscation:** When dealing with sensitive object methods such as `Run`, `ShellExecute`, and `Create` in VBA macros, the `CallByName` method can be useful in executing a method by specifying its name through a string. By combining it with other forms of string obfuscation, the called method can be further obfuscated. The macro depicted in Figure 7(b) is the obfuscated version of the code shown in Figure 7(a), where the object's called method can be seen as obfuscated.

<pre>If IsError(found) Then msg = msg & rowMsg & " " & cellAddress & " - Invalid, select from list." & vbCrLf End If</pre>	<pre>lw = "" Set obj = GetObject("new:C08A" + lw + "FD90-F" + lw + "2A1-11D1-845" + lw + "5-00A0C91F3880")</pre>
(a)	(b)

Figure 7. Call obfuscation example. (a) Example of macro before call obfuscation; (b) Example of macro after call obfuscation.

- **Logical Obfuscation:** The concept of logical obfuscation is about adding dummy code like procedure calls and branch decisions, which complicate the macro's logic. In certain cases, attackers conceal a few lines of malicious statements among large amounts of normal macro code, making it problematic to analyze malicious macros - similar to searching for a needle in a haystack. Without comprehending the program's semantics, it is impossible to determine if the code is logically obfuscated.

3. Related Work

Although there are some tools available for analyzing Office macros in industry [11,17], researches of detecting malicious office macros are still scarce. Most such researches are machine-learning based and some recent related works are as follows.

Nir Nissim et al [14] extracted structural features from docx documents and employed machine learning to detect malicious docx documents. Assisted by experts, with active learning, it achieved a 94.44% detection rate and a low false alarm rate of 0.19%. The main limitation of the research is that it only applies to docx documents. Kim et al[8] investigated VBA macro obfuscation in Office, wherein they categorized VBA macro obfuscation techniques into four types and introduced a feature set for detecting obfuscation effectively. However, they clarified that their study is a way to detect obfuscation and not to detect malicious macros. Where obfuscation techniques are implemented to protect intellectual property rights, it is imperative to differentiate between malicious and obfuscated macro detection. Failure to do so will result in an elevated occurrence of false alarms when utilizing obfuscation characteristics to identify malicious macros. Vasilios et al[9] analyzed a substantial body of office documents containing VBA macros and utilized 40 suspicious keywords as features to identify malicious macros. The machine learning model, Random Forest, achieved an accuracy of 97.5%. However, it appears that these features do not adequately cover certain malicious macros that utilize new keywords, let alone unknown ones in the future.

Other than VBA source code, Simon et al. [15] concentrated on the compiled version of VBA source code, known as p-code, within office documents. They adopted 2-gram features and obfuscation features to identify malicious p-code based office documents. As XL4 macros in Excel have become significant attack vectors, Ruaro et al [16] introduce SYMBEXCEL which uses symbolic execution to automatically de-obfuscate and analyze these macros, enabling users to extract Indicators of Compromise (IoCs) and other vital forensics information with reliability.

4. Machine learning Method with combined features

4.1. Data collection and Pre-processing

To compare with previous studies, we adopt the same dataset from Vasilios [9], which comprises 2968 macro-based benign office documents and 15570 macro-based malicious office documents. The benign samples are collected from 726 governmental and 1010 educational sites across different countries and institutions. Meanwhile, the malicious samples originate from three renowned repositories, AppAny, Virusign, and Malshare.

In the pre-processing phase, we extract the VBA macro from the available samples and remove those that do not satisfy our requirements as shown in the part immediately following. Subsequently, we label the remaining data for training and testing. To extract macros from office documents automatically, we utilize Oletools, an open-source python-based tool, which has also been used for suspicious keywords extraction.

- Malicious samples disarmed

After conducting preliminary analysis of the macros in this dataset, we discover that a number of malicious macros contain the message "Macro code was removed by Symantec Disarm". This suggests that the malicious part of the macros was removed; Therefore, it may not be appropriate to classify them as malicious. To confirm this, the malicious samples were re-scanned by ClamAV antivirus software. After scanning, 14,064 samples were identified as malicious, and the remaining 1,506 samples were removed from the malicious dataset.

- Excel 4.0 macro samples

As Excel 4.0 macros differ significantly from VBA macros, detecting Excel 4.0 macros falls outside the scope of our experiment. Therefore, we have excluded pure Excel 4.0 macros to achieve more accurate results, and retained mixed samples which contain both VBA and Excel 4.0 macros. We acquire 171 samples with malicious Excel 4.0 macros, and find no Excel 4.0 macros in benign samples.

- Samples unable to parse with oletools
- It is noteworthy that certain malicious samples are not parsed sucessfully with the use of Oletools. These samples appear to be intentionally designed to exploit Oletools' weaknesses and escape detection. This highlights the strategic maneuvers and countermeasures in the ongoing battle between attackers and defenders. Totally 155 malicious samples could not be parsed by Oletools. However, the tool is able to parse all benign samples.
- Samples without macros
- We still find some samples with no macros inside and abandon them.

After pre-processing, our final dataset contains 2939 benign samples and 13734 malicious samples containing VBA macros. Table 1 presents the breakdown of document types and quantities of benign and malicious samples. Excel documents are the primary format with benign macros, while word documents are favoured by attackers with malicious intention. This is because word documents are more seductive compared to excel documents. However, certain file types, including Office Publisher and encrypted documents with CDFV2, may not be recognized and are therefore categorized as type "others".

Table 1. Details of purified data set from Vasilios.

label	benign	malicious
number	2939	13734
Details of document type	doc 644, docx 26, xls 1607, xlsx 662	doc 11025, docx 1340, xls 1012, xlsx 240, ppt 5, pptx 5, others 107

It is common knowledge that malicious samples tend to mutate over time. Therefore, a suitable detection model must be able to adapt to this change. To assess the model's robustness, we obtained a batch of new malicious documents containing macros from VirusTotal with a submission date after Vasilios[9] published. All these samples were identified as malicious by over 10 antivirus software in VirusTotal. We gathered 2885 malicious samples with a time span from 2021-2-25 to 2022-5-20, where more information about this dataset is shown in Table 2. The obfuscation tag is acquired from Virustotal which shows us the basic information about obfuscation trends. For convenience, we tag the former dataset as Dataset1 and the latter dataset as Dataset2.

Table 2. New malicious samples collected from Virustotal.

label	malicious
number	2885
Details of document type	doc 1537, docx 58, xls 1193, xlsx 16, ppt 13, others 68
Details of obfuscation	Obfuscated 1358, non-obfuscated 1527

4.2. Feature selection

4.2.1. Obfuscation features

It is important to clarify that our focus is on detecting malicious macros, not obfuscated ones. While we list six categories of obfuscation techniques in section 2, some of these obfuscations are often essential for attackers in malicious macros. For example, malicious commands. can only be obfuscated by split, encoding or embedding obfuscation. If the attacker also wants to conceal suspicious method calls, call obfuscation is essential. To detect malicious macros, it is unnecessary and difficult to extract features from all types of obfuscation. Instead, we focus on commonly used obfuscations by attackers, including split, encoding, and call obfuscation.

When extracting obfuscation features, the following principles are followed: 1) they should be clear and simple, requiring no complex analysis of macro semantics and 2) they must be resilient against adversarial attacks. For instance, previous studies have regarded the length of comments in

macros as an obfuscation feature [8,18]. However, since adding comments to macros is effortless for attackers to create adversarial samples, we contend that this feature lacks robustness and have removed comments prior to feature extraction. Symbols for convenience in understanding our proposed features are introduced in Table 3. The features and their descriptions are presented in Table 4.

Table 3. Symbols.

Symbols	Description
$L = \langle l_1, l_2, \dots, l_n \rangle$	l_i represents a line in macro's procedures, L is the vector of all lines in macro.
$P = \langle p_1, p_2, \dots, p_m \rangle$	p_i represents a procedure in macro, P is the vector of all procedures in macro.
count_concatenation	count the number of concatenation symbols include "+" and "=".
count_arithmetic	count the number of arithmetic operators include "+", "-", "*", and "/".
count_parentheses	count the number of left parentheses adjacent to a word.
count_assignment	count the number of "=".
count_strings	count the number of strings.
length	calculate the length
max	calculate the maximum

Table 4. Features to detect malicious macro.

Feature name	Description
F1	$\max\{\text{length}(l_i)\}, i \text{ from } 1 \text{ to } n$
F2	$\max\{\text{count_concatenation}(l_i)\}, i \text{ from } 1 \text{ to } n$
F3	$\max\{\text{count_arithmetic}(l_i)\}, i \text{ from } 1 \text{ to } n$
F4	$\max\{\text{count_parentheses}(l_i)\}, i \text{ from } 1 \text{ to } n$
F5	$\max\{\text{count_strings}(l_i)\}, i \text{ from } 1 \text{ to } n$
F6	$\max\{\text{count_concatenation}(p_i)\}, i \text{ from } 1 \text{ to } m$
F7	$\max\{\text{count_arithmetic}(p_i)\}, i \text{ from } 1 \text{ to } m$
F8	$\max\{\text{count_parentheses}(p_i)\}, i \text{ from } 1 \text{ to } m$
F9	$\max\{\text{count_strings}(p_i)\}, i \text{ from } 1 \text{ to } m$
F10	$\max\{\text{count_assignment}(p_i)\}, i \text{ from } 1 \text{ to } m$
F11	$\text{length}(P)$
F12	$\text{length}(L)$
F13	the number of occurrences of function CallByName
F14	the number of occurrences of each conversion function, math function, string function
F15	the number of occurrences of each suspicious keywords

Detecting coding obfuscation and split obfuscation in a single line can be achieved through the evaluation of F1-F5 values, which indicates the complexity of the line. F1 represents the maximum line length, while F2 and F5 represent the maximum number of split operators and strings within the line. When split obfuscation has been used, the values of F2 and F5 in malicious macros are more

likely to be greater than that in benign ones. Therefore, we utilize F2 and F5 to identify split obfuscation. F3 represents the maximum number of arithmetic operators that occur in a line. Figure 5 illustrates that arithmetic operations are sometimes used in coding obfuscation to hide its attacking intention. F4 represents the maximum number of left parentheses adjacent to a word within a line. The left parenthesis adjacent to a word indicates a procedure call, so this feature signifies the number of procedure calls in a line, which includes conversion functions, mathematical functions, and string functions utilized in coding obfuscation.

To identify obfuscation within a singular procedure, the use of F1-F5 is ineffective if an attacker disassembles the obfuscated code in a line across the procedure. To address this issue, F6-F10 have been implemented. Features F6-F9 are similar to F2-F5, but their scope has been changed from a line to a procedure. These features may reflects both split obfuscation and coding obfuscation within a procedure. Additionally, F10 identifies the maximum number of assignment operators ("=") present in procedures. If a line is disassembled into multiple lines, more assignment operators would be used.

If an attacker spreads the obfuscated code into numerous procedures and attempts to make every procedure's features F1-F10 as normal as possible, more procedures and lines will be required. This is comparable to adding a spoon of salt into a large pool of water, more water will be needed to dilute the saltiness of the water. We use F11 and F12 as indicators of this type of dilution. F11 represents the number of procedures in a macro, while F12 represents the total number of lines in a macro.

In addition to the statistical features mentioned above, we have also propose several frequently used obfuscated features. F13 illustrates the frequency of the CallByName function used in macros, which may provide insight into call obfuscation. F14 is actually a collection of sub features showing the number of occurrences of each conversion function, math function, and string function, such as Asc, Chr, Mid, Join, InStr, Replace, Right, StrConv, Abs, Atn, Hex, Oct. A total of 64 functions are included in F14. We utilize this feature to identify obfuscation that may not be detected from statistical features, as these functions are commonly used in split and encoding methods. For instance, while features F1-F12 may not be capable of identifying obfuscation statistically insignificant, feature F14 reveals the called obfuscated functions.

4.2.2. Suspicious keywords features

Malicious macros are usually more inclined to use specific functions or strings to execute commands than normal ones. For instance, benign macros typically do not utilize the "Shell" function to execute programs, whereas attackers frequently employ this method. Table 5 indicates the keywords frequently used in malicious macros. F15 identifies 46 of these keywords as suspicious. It's worth mentioning that the keywords selected for detecting malicious macros are not comprehensive due to their complexity [11]. Only the most commonly used suspicious keywords have been included.

Table 5. Features of suspicion to detect malicious macro.

Keywords	Description
Auto_Open*, AutoOpen*, Document_Open*, Workbook_Open*, Document_Close*	Procedure names will be executed automatically upon opening or closing the document
CreateObject*, GetObject, Wscript.Shell*, Shell.Application*	Methods and parameters to obtain the key object capable of executing commands
Shell*, Run*, Exec, Create, ShellExecute*	Methods used to execute a command or launch a program

CreateProcessA, CreateThread, CreateUserThread, VirtualAlloc, VirtualAllocEx, RtlMoveMemory, WriteProcessMemory, VirtualProtect, SetContextThread, QueueApcThread, WriteVirtualMemory, Print*, FileCopy*, Open*, Write*, Output*, SaveToFile*, CreateTextFile*, Kill*, Binary*	External functions, when imported from kernel32.dll, can be used to create a process or thread, operating memory
cmd.exe, powershell.exe, vbhide*	Command line tools and suspicious parameters
StartupPath, Environ*, Windows*, ShowWindow*, dde*, Lib*, ExecuteExcel4Macro*, System*, Virtual*	Other keywords related to startup Path, environment variables, program windows, dde, function reference of DLL, Execution of Excel 4 macro, virtualization

Keywords with a asterisk are used in [9].

5. Evaluation

After extracting the features from both Dataset1 and Dataset2, we train four classical machine learning models to detect malicious macros, including Random Forest (RF), Multi-layer Perceptron (MLP), Support Vector Machine (SVM) and K-Nearest Neighbors (KNN). The configuration parameters for each model are listed in Table 6. For model SVM and KNN, their input has been standardized before testing and training.

Table 6. Configuration of models.

Model	Parameter
RF	n_estimators=100
MLP	hidden_layer_sizes=(150,), max_iter=500
SVM	kernel='rbf'
KNN	n_neighbors=3

We initially train and test the aforementioned models on Dataset1 with varying feature selection, including solely obfuscation features (F1-F14), suspicious keyword features (F15), and combined features (F1-F15). Our experiments were conducted on a Huawei 2488V5 server with 2 Xeon (R) Gold 5118 CPUs, 64 GB system memory, and the CentOS 8 Stream operating system. We assess the effectiveness of the trained classifiers using standard classification metrics such as precision, recall, accuracy, and F1-score. To have a clear indication of the false positives for benign samples, considering the imbalance of malicious and benign samples in Dataset1, we also incorporated false alarm rate (FAR) into the metrics. Table 7 shows the performance of different models, which comes from 5-fold cross-validation on Dataset1. To compare with the work presented in [9], we add the best results reported in their study to the final row of Table 7.

Table 7. Performance results of models on Dataset1.

Model	Feature Selection	FAR	Precision	Recall	Accuracy	F1-Score
RF	F1-F14	0.083	0.982	0.994	0.981	0.988
	F15	0.012	0.997	0.995	0.993	0.996
	F1-F15	0.015	0.997	0.997	0.994	0.997
MLP	F1-F14	0.138	0.971	0.979	0.958	0.975
	F15	0.013	0.997	0.995	0.994	0.996

	F1-F15	0.035	0.993	0.994	0.989	0.993
	F1-F14	0.203	0.957	0.961	0.932	0.959
SVM	F15	0.022	0.995	0.987	0.986	0.991
	F1-F15	0.028	0.994	0.992	0.988	0.993
	F1-F14	0.092	0.980	0.984	0.971	0.982
KNN	F15	0.020	0.996	0.992	0.990	0.994
	F1-F15	0.028	0.994	0.992	0.988	0.993
RF	Ref [9]	—	0.993	0.976	0.975	0.985

After conducting cross-validation experiments on Dataset1, we utilize all the samples within Dataset1 to train our four machine learning classifiers. Subsequently, we carry out the same experiment on Dataset2, which consists only malicious samples. This time we adopt precision as the metric for Dataset2. Table 8 shows the results.

Table 8. Performance results of models on Dataset2.

Model	Feature Selection	Precision
	F1-F14	0.923
RF	F15	0.856
	F1-F15	0.953
	F1-F14	0.739
MLP	F15	0.793
	F1-F15	0.907
	F1-F14	0.799
SVM	F15	0.788
	F1-F15	0.817
	F1-F14	0.792
KNN	F15	0.757
	F1-F15	0.735

From Table 7, it is evident that RF classifier with F1-F15 achieves the highest F1-score among all models. Furthermore, detecting by suspicious keywords (F15) always enjoys a better detection performance than obfuscation features (F1-F14). At first glance, introducing obfuscation features on the basis of suspicious keywords does not significantly improve detection performance and even slightly worsen for MLP and KNN models. Nonetheless, from table 8, it is evident that there is a obvious decline in performance for models using just suspicious keywords as features. Interestingly, however, with the combined features, the random forest model has a precision of 0.953 on the 2885 new malicious samples. This indicates that the integration of the obfuscation features and suspicious keywords enhances the model's resilience.

Comparing with work [9], our RF classifier trained by suspicious keywords (F15) has a higher recall and F1-Score. This is likely due to the inclusion of new suspicious keywords such as GetObject and external functions in our features, which can detect more malicious samples.

Table 9 shows the top 20 features of the RF classifier, which use combined features to detect malicious samples in Dataset2. It reveals that suspicious keywords only make up half of the prominent features. Additionally, ten obfuscation features, including four statistic features, demonstrate their significance in detecting malicious macros. This highlights the potential for complementarity between both feature types in improving detection performance. We further analyze the proportion of benign and malicious for samples containing keywords from F14 and F15. Table 10 demonstrates that samples that contain these keywords are more likely to be malicious samples.

Table 9. The top 20 prominent features and their type.

Index	Feature Name	Feature Type
1	F15:CreateObject	Suspicious Keywords
2	F15:Document_Open	Suspicious Keywords
3	F15:Shell	Suspicious Keywords
4	F15:GetObject	Suspicious Keywords
5	F15:Lib	Suspicious Keywords
6	F15:AutoOpen	Suspicious Keywords
7	F15:Auto_Open	Suspicious Keywords
8	F15:StartupPath	Suspicious Keywords
9	F11:length(P)	Obfuscation
10	F3:max{count_arithmetic(l_i)}	Obfuscation
11	F14:Chr	Obfuscation
12	F14:Asc	Obfuscation
13	F14:UCase	Obfuscation
14	F7:max{count_arithmetic(p_i)}	Obfuscation
15	F5:max{count_strings(l_i)}	Obfuscation
16	F14:Left	Obfuscation
17	F15:Open	Suspicious Keywords
18	F14:Abs	Obfuscation
19	F14:Split	Obfuscation
20	F15:System	Suspicious Keywords

Table 10. Proportion of benign and malicious for samples containing keywords from F14 and F15.

Index	Feature Name	Proportion of Benign samples	Proportion of malicious samples
1	F15:CreateObject	2.76%	97.24%
2	F15:Document_Open	0.41%	99.59%
3	F15:Shell	0.29%	99.71%
4	F15:GetObject	0.07%	99.93%
5	F15:Lib	1.72%	98.28%
6	F15:AutoOpen	0.12%	99.88%
7	F15:Auto_Open	9.62%	90.38%
8	F15:StartupPath	0.00%	100.00%
9	F14:Chr	2.31%	97.69%
10	F14:Asc	1.88%	98.12%
11	F14:UCase	34.20%	65.80%
12	F14:Left	10.94%	89.06%
13	F15:Open	5.75%	94.25%
14	F14:Abs	27.32%	72.68%
15	F14:Split	3.45%	96.55%
16	F15:System	12.27%	87.73%

We further investigate why there are still some samples in dataset2 that can not be detected by the RF classifier with combined features. It’s interesting to find that some of these undetected samples can be detected by RF classifier with feature selection F1-F14 or F15 alone. A schematic diagram has been included to illustrate sample detection using the RF classifier on Dataset2 (see Figure 8). The samples successfully detected by the corresponding classifier are represented by the points inside of the circle, while the points outside the circle depict the undetected samples. It is apparent that each classifier can identify part of samples. We believe that the combination of features may weak some critical characteristics which seem obvious and easy to learn from the only one type of features. These

features are subsequently placed at lower level nodes in many of the decision trees within the random forest. This may change the classification results for some samples that are correctly classified in a single type of feature.

Based on the above experiments and analyses, we can enhance our detection ability by integrating multiple classifiers. Table 11 illustrates the precision of Dataset2 with different ensemble of classifiers. If we combine the RF classifier with F15 and the RF classifier with F1-F15, the precision would be 0.969. When we join all three RF classifiers to detect samples on Dataset2, we can achieve a precision of 0.980. By the way, most of the samples that have not yet been detected are malicious macros based on P-Code, which is out of the scope in this paper. It should be noted that this basic ensemble may result in increased false positives. Table 6 shows that the classifier with F1-F14 consistently has a higher rate of false positives comparing with other classifiers. Therefore, it depends by specific scenarios when selecting ensemble learning models and combining these classifiers.

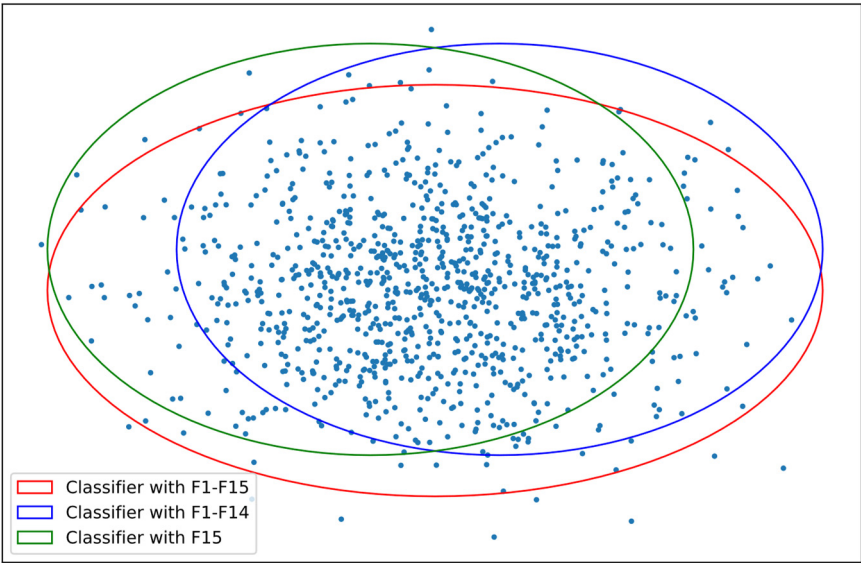


Figure 8. Sample detection diagram of RF classifier.

Table 11. Precision with different ensemble of classifiers.

Ensemble of classifier	Precision
RF classifier with F1-F14 and RF classifier with F15	0.979
RF classifier with F1-F14 and RF classifier with F1-F15	0.974
RF classifier with F15 and RF classifier with F1-F15	0.969
All three Classifiers	0.980

6. Discussion and Conclusion

In this paper, we propose a novel approach to improve the detection of malicious macros by combining obfuscation features and suspicious keywords. We extract 78 obfuscation features and 46 suspicious keywords. Then we conduct experiments on two datasets. Dataset1 consists of 2939 benign samples and 13734 malicious samples, while Dataset2 contains 2885 new samples that has been reported after the publication date of Dataset1. The initial experiment conducted on Dataset1 indicates that the highest performing classifier is RF with combined features, obtaining an F1-score of 0.997. When utilizing the model trained on Dataset1 to identify forthcoming samples on Dataset2, the precision of RF classifier with suspicious keywords declines significantly from 0.996 to 0.856. In contrast, the RF classifier with combined features is capable of detecting 95.3% of malicious samples, demonstrating that the combined features can sustain model robustness with high detection rate. The top 20 prominent features show that both obfuscated features and suspicious keywords play an important role in detecting malicious macros.

We analyze both obfuscation features and suspicious keywords to identify malicious macros, assuming that malicious macros are always obfuscated and specified functions or strings are integral. This enables our model to identify malicious samples using keywords that were previously unknown to the community. If these new keywords can be identified by experts from the samples and subsequently used to retrain the model with additional features and samples, similar to the approach taken in [14], the model will maintain a greater level of robustness. However, this does not mean that attackers cannot bypass our model. At the moment, our obfuscation features do not cover embedding obfuscation because it is challenging to differentiate between benign macros and malicious macros. If an attacker combine embedding obfuscation and newly discovered attack techniques, it is highly possible that our classifier would be bypassed. This issue remains for our future work.

Data Availability Statement: The data and source code presented in this study are available in https://gitee.com/chensheng101/macro_fc.

References

1. <https://www.microsoft.com/en-us/security/blog/2018/09/12/office-vba-amsi-parting-the-veil-on-malicious-macros/>
2. Matt Miller. Trends and Challenges in the Vulnerability Mitigation Landscape. <https://www.usenix.org/conference/woot19/presentation/miller>
3. <https://learn.microsoft.com/en-us/deployoffice/security/internet-macros-blocked>
4. <https://www.csoonline.com/article/573293/attacks-using-office-macros-decline-in-wake-of-microsoft-action.html>
5. <https://www.fortinet.com/blog/threat-research/are-internet-macros-dead-or-alive>
6. <https://thehackernews.com/2022/12/apt-hackers-turn-to-malicious-excel-add.html>
7. <https://thehackernews.com/2022/07/microsoft-resumes-blocking-office-vba.html>
8. Kim, Sangwoo, et al. "Obfuscated VBA macro detection using machine learning." 2018 48th annual IEEE/IFIP international conference on dependable systems and networks (dsn). IEEE, 2018.
9. Koutsokostas, Vasilios, et al. "Invoice# 31415 attached: Automated analysis of malicious Microsoft Office documents." Computers & Security 114 (2022): 102582.
10. Advanced VBA macros: bypassing olevba static analyses with 0 hits. <https://www.certego.net/blog/advanced-vba-macros-bypassing-olevba-static-analyses/>
11. <https://github.com/decalage2/oletools/wiki/olevba>
12. <https://learn.microsoft.com/en-us/office/vba/api/overview/>
13. Vasilios Koutsokostas, Nikolaos Lykousas, Gabriele Orazi, Theodoros Apostolopoulos, Amrita Ghosal, Fran Casino, Mauro Conti, & Constantinos Patsakis. (2021). Malicious MS Office documents dataset. <https://doi.org/10.5281/zenodo.4559436>
14. N. Nissim, A. Cohen and Y. Elovici, "ALDOX: Detection of Unknown Malicious Microsoft Office Documents Using Designated Active Learning Methods Based on New Structural Feature Extraction Methodology," in IEEE Transactions on Information Forensics and Security, vol. 12, no. 3, pp. 631-646.
15. S. HUNEAULT-LEBLANC and C. TALHI, "P-Code Based Classification to Detect Malicious VBA Macro," 2020 International Symposium on Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, 2020, pp. 1-6
16. N. Ruaro, F. Pagani, S. Ortolani, C. Kruegel and G. Vigna, "SYMBEXCEL: Automated Analysis and Understanding of Malicious Excel 4.0 Macros," 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 1066-1081
17. ViperMonkey. <https://github.com/decalage2/ViperMonkey>
18. Aebbersold S, Kryszczuk K, Paganoni S, et al. Detecting obfuscated javascripts using machine learning[C]//ICIMP 2016 the Eleventh International Conference on Internet Monitoring and Protection, Valencia, Spain, 22-26 May 2016. Curran Associates, 2016, 1: 11-17.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.