Macro Malware Detection using Machine Learning Techniques
A New Approach

Sergio De los Santos and José Torres
ElevenPaths, Telefónica Digital Cyber Security Unit, Madrid, Spain
{ssantos, jose.torres}@11paths.com

Keywords: Macro, Malware, Office, Classification, Machine Learning.

Abstract: A malware macro (also called "macro virus") is the code that exploits the macro functionality of office documents (especially Microsoft Office's Excel and Word) to carry out malicious action against the systems of the victims that open the file. This type of malware was very popular during the late 90s and early 2000s. After its rise when it was created as a propagation method of other malware in 2014, macro viruses continue posing a threat to the user that is far from being controlled. This paper studies the possibility of improving macro malware detection via machine learning techniques applied to the properties of the code.

1 INTRODUCTION

Originally macros add extra functionalities to documents, providing them with dynamic properties that allow, for example, to perform actions on a set of cells in an Excel document or embed multimedia objects in Word files. But by the late 90s, they started to become an attack vector used by malware creators to execute code in systems. Attackers would program macros that extended their functionality for the execution of malicious actions on the system, such as downloading and running executables.

"Melissa" (Wikipedia, n.d.) was one of the most recognized and harmful worms, in March, 1999. The way of spreading this type of malware traditionally has been (and still is) email. The victim receives an email with an attachment and when opens it, the internal macro is executed and infects the operative system. In recent years, following the improvements introduced by Microsoft in the Office package to prevent the automatic execution of macros, this type of malware has lost relevance. The existence of other more direct methods that did not depend on the configuration of the Office system (e.g. exploiting vulnerabilities) caused that, for a while, this formula lost popularity. However, since 2014, Microsoft is warning of a significant rebound (Pornasdoro, 2014) in the use of macro malware, this time as a spread method or a way for downloading other malware.

Since 2104 and during 2015 (MMPC, 2015), macro viruses have been used to spread ransomware malware or banking trojans, quite successfully for attackers despite the countermeasures and security improvements. Therefore, since more than 15 years after its appearance, macro malware remains a threat, and mechanisms to detect these attacks are still necessary.

The purpose of this paper is to study the behavior of attackers when creating malicious macros and their functioning. Moreover, it wants to demonstrate if detecting, analyzing and using the most common methods of malware programming and obfuscation may facilitate the correct and automatic classification and distinction of documents containing legitimate macros, from those with malicious macros. While it is assumed that this is a specific task of antivirus systems, this study does not intend to replace them, but to describe a different approach based on parameters other than signatures or heuristics to complement detection through these traditional systems and allow a more effective identification in another layer and with other means such as machine learning.

The rest of the paper is organized as follows: section two analyses the technical structure of these documents and a historical introduction; section three shows the background both in terms of previous related studies and existing tools to introduce the problem; section four describes and develops the proposal; section five presents the

Santos, S. and Torres, J.
Macro Malware Detection using Machine Learning Techniques - A New Approach.
DOI: 10.5220/0006132202950302
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 295-302
ISBN: 978-989-758-209-7

results of this work to finally describe the conclusions and possible future work in the last section.

2 BACKGROUND

Visual Basic for Applications (Wikipedia, n.d.) is the language used to create macros in Office. It appeared in 1993, and its latest version dates from 2013. It is related to Visual Basic, in the sense that it needs its engine to run, but it is not independent: it must run within another application that contains the code, and interact with other applications through OLE Automation objects (a Microsoft internal CPI).
VBA is compiled into P-Code (also used in Visual Basic). This is a proprietary system from Microsoft that allows its decompilation to the original format in which the code was written. Once compiled, it is stored in the corresponding Office document as a separate flow in an OLE or COM object.
Since 2007 there are two very different formats of Office documents, and depending on the version of the Office format used, this object can be found embedded in the document or as a separate file. The different formats are:

- Based on Microsoft formats prior to 2007 with .doc or .xls extensions ("classic" format). Formats prior to 2007 are actually an OLE object in themselves.
- Based on Open XML formats (Microsoft, s.f.) after 2007 with .docx or .xlsx extensions for example. These formats are actually ZIP files, which contain the same COM object as a macro.

COM or OLE objects used by Microsoft to store macros are specifically OLE objects with the structure "Office VBA File Format".

3 STATE OF THE ART

From the macro malware analysis standpoint, on the Internet we can find numerous specific analysis about malware that uses different techniques that maximize the chances of VBA to get control of the system and execute code. In the antivirus industry, numerous patents have been created to control this type of malware, such as (Ko, 2004), that describes how to extract the macro in a document, analyses the flow and operations of the code, compares against a database previously categorized and issues a verdict. Improving the previous approach, since the late 90 new malware detection techniques appear based on program behavior analysis, such as (Chi, 2006), patented by Symantec in 2006. As a third approach, we can categorize those techniques, for example (Shipp, 2009), consisting of a more thorough analysis of the code itself through the use of statistics, but always limited to the morphological aspect, that is, comments, character frequency, names of variables and functions, etc.

In addition to the above, a new and different approach can be taken into account, based on the use of machine learning techniques to detect malware. In this case, most of the existing literature comes from academia and is considerably less extensive than that addressed by the aforementioned approaches. For example, in (Nissim, et al., 2015) Nir Nissim et al use a methodology they have named Active Learning in which they use machine learning techniques in order to, from Open XML formats (.docx extension), extract features from the document that are external to the code using a system called SFEM, and that when combined with their learning system ALDOCX, help identify malware on office documents. The extraction of SFEM features is based on obtaining internal paths of the ZIP composing the document. This system is restricted only to new formats based in Office's XML, and it needs the full document to work properly, including text or relevant content, which may violate privacy if information control during analysis is not strict enough.

Furthermore, in (Schreck, et al., 2013) Schreck et al presented in 2013 another approach which they called Binary Instrumentation System for Secure Analysis of Malicious Documents, which sought to distinguish malicious documents extracting the malicious malware payload and identifying the exploited vulnerabilities. However, it only worked for classifiying classic Microsoft formats (.doc extension).
The technique and framework presented in this research is able to work with both classic format and Open XML-based documents. In addition, it relies primarily on the characteristics of the VBA project code and other metadata of the file, but it completely detaches from the contents of the document or any aspect that allows establishing a connection with a particular document. The use of metadata is limited, focusing the machine learning on the code features that define them at semantic level. Moreover, as we will analyze in further sections, we potentiate the selection of features similar to that used by Schreck, et al., automating it and making it dynamic in time.

4 DESCRIPTION OF THE PROPOSAL

The research proposal, therefore, is based on the application of machine learning techniques, in this case classification and supervised learning techniques, to determine whether by creating a specialized classifier it is possible to become more effective than the solutions most commonly used and known nowadays, which basically are limited to antivirus engines.

4.1 Identification and Collection of Samples

Since the goal is to build a classification system, the fundamental initial process will be the collection of samples. This exercise should help obtain a representation of the universe to be studied as realistic as possible. For this research, we have taken samples from Word office documents in .doc, .docx format as well as in .docm from different sources and created at different times, although with a significant percentage of recent and current samples (about 80% of the samples collected were created in 2015 or 2016). The samples were collected from diferent sources as for example email accounts that usually receive spam or malware attached, repositories of public malware (malwr.com, contagiodump, etc.) and repositories of documents in general (P2P networks, search engines and repositories of public documents, etc.)

We recovered a total of 1,671 office documents. For the sample identification and classification, we created two different classes or sets:

- Goodware: Word files with macros from more trusted sites (document repositories in domains of public entities, universities, etc.) and without a heuristic pattern detected that could be described as suspicious. For example, obfuscated strings, the presence of suspicious calls to API, etc. In addition to the absence of suspicious indicators, we used multiple antivirus engines to validate that they were not detected as malware.
- Malware: Samples that, analyzed with more than three different antivirus engines, were detected as malware by at least three of these engines, regardless of their characteristics.

Considering the detection by three or more engines as “a threshold” for malware is arbitrary, although accepted by literature in general. Choosing a good detection "threshold" is a complex exercise that still has not been solved or standardized, but that can make the result of a research based on classification to throw different results depending on whether it is malware or goodware, two terms that are not always properly distinguished even by antivirus engines.

4.2 Analysis of Samples and Coding of Features

To create a classifier, it is necessary to decide which features will be taken from the samples and how they will be coded. These features will serve as predictors that will form the input vector of the classifier to be built. In this phase of the study, the features are not final yet, but respond to the bulk of all those features that have been obtained from samples using the available tools.
For example, some features may seem to have no impact on the result of the classification, others however may have a direct correlation, and others may be even correlated with each other and in turn be decisive for the outcome. In the following charts (Figures 1 and 2), we can see how the linear adjustment of the ratio shows a certain inverse relation between the size of samples (in megabytes) and the number of detections by antivirus engines.

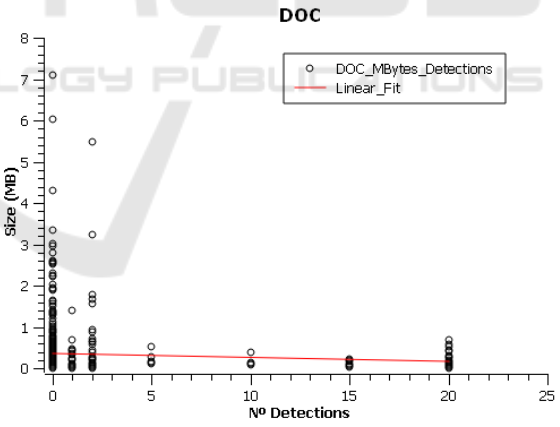

Figure 1: Document size (MB) vs Number of detections.

However, in the case of size of the VBA project where macros are stored, the linear adjustment reveals that the relation is direct, that is, the larger the size of the VBA project, implies a greater number of antivirus engines that would detect the sample. This may be because, usually, in the case of Word documents, macros are typically not too complex, so the generated VBA code is not complex either.

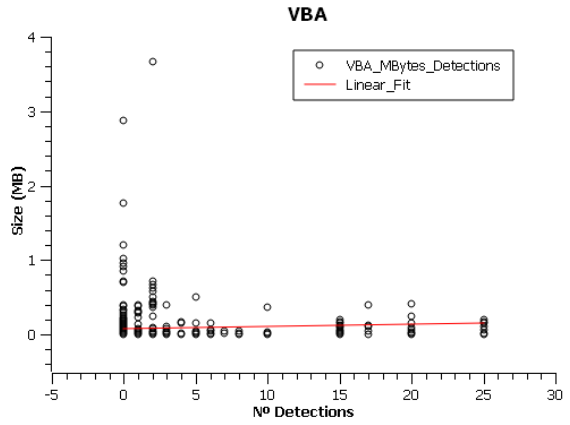

Figure 2: VBA Project size (MB) vs Number of detection.

However, in the case of malicious macros, the amount of code is higher, since usually many functions and procedures are needed to perform such actions. In any case, this continues to be an assumption based on the observation of the researcher that cannot actually be demonstrated without some kind mechanism such as the one presented in this study.

Thus, thanks to our implementation of an automatic selection of characteristics (ASC) based in PCA, the outflow of the classifier (once validated by analysts) serves as feedback and becomes part of the training set. The continuous arrival of new samples to this training set, will lead to the emergence of new features and to changes in the weighting performed by the classifier. These changes must be automatically detected by the ASC depending on the quality of the results obtained by the classifier.

This research relies heavily on information provided by the decompiled code of the macro, although additionally we have included heuristic characteristics used by the Python framework python-oletools. This way, the input characteristics vector has been reduced to a minimum size appropriate to the needs of the classifier, and it allows sufficient flexibility in terms of automatic selection of characteristics for the results obtained to be significantly improved. The proportion of characteristics directly related to the decompiled macro code that make up the vector is 76%.

The coding of the features in the vector is binary, since the load of the classifier is relieved by reducing the search space and therefore increasing its performance in terms of time and amount of computation.

Currently we work with four classification algorithms: Binary Decision Trees, Support Vector Machines, Random forest and Neural Networks, although the system as a whole has been designed so that the used classifier is an interchangeable piece, both at algorithm level and version of the classifier itself. Therefore, depending on the results, we will be able to quickly and easily change between any of the three algorithms implemented without the functioning of the system being affected. Moreover, new algorithms may be added to the existing ones simply respecting the modular structure of the system with which the previous ones were added, which is a relatively simple process.

In addition to the classifier, we have developed a framework that acts as a wrapper, adding a layer of high-level functionalities, such as the capacity to relate malware or add new samples and analyze them. With this, among other things, the classifier is supplied automatically with different sample sources that are dumped on an unified deposit for their subsequent processing. Thus, the number of samples will progressively increase, taking also into account that the system itself is a source of incorporation of samples that are uploaded by users through the platform that has been developed so that analysts, among others, can interact with the system in a more user-friendly way.

Among the features used to form the vector, we have taken into account specific characteristics of the code from macros as well as other features of the VBA project and the document itself. For example, one of the code's characteristics taken into account could be the use of words reserved in VBA related to macros auto-execution (AutoExec, AutoOpen, Document Open, etc.), or invocations to typical access libraries to APIs of the system that allow a higher functionality in macros. For the implementation of extractions of such characteristics, we have used the Python library OleVBA.

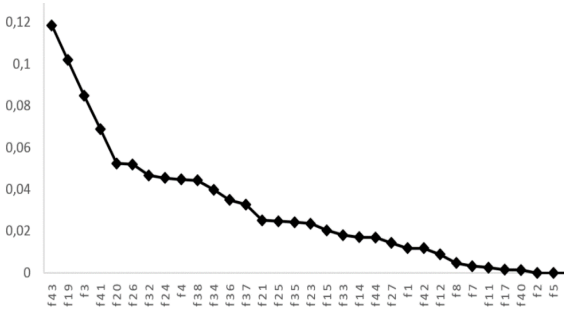

Figure 3: Graphical representation of weighting of characteristics.

On the other hand, an example of a feature both of the VBA project and of the Word document itself could be the time difference between its creation and the last modification, or the number of macros in a document.

For extraction and analysis of these characteristics, a specific program is created for our needs. Based on the Python library OleFile, it compensates the scarcities found in OleVBA, complements it, and also directly produces the vector that characterizes it for each input sample. This vector is composed of 45 bits in total, but not all features are purely binary (some have been discretized). This means that 45 bits do not necessarily involve 45 different characteristics, but some monopolize up to 4 bits to characterize the sample in a more granular way. For example, macros sizes are discretized that way, grouping them by ranges.

Definitely, as a result of all the above, we obtain a base vector (BV), which contains all significant features that have been commonly extracted from all types of referred documents (doc, docx and docm), and that defines them anonymously as input for the classifier. Since this is a first version of the system, we should mention that this vector can vary over time, either by adding new features that can be drawn from the documents already analyzed, or by the inclusion of new documents, such as Excel, PowerPoint or PDF that provide new features themselves.

4.3 Feature Selection

Weka was used at the earliest stages of the study in order to make a first approach that served as a quick method of validation of the functionalities for a later debugging based on successive iterations. Given the structure of the extracted information as how it is reflected in the vector (as binary predictors), the type of classifier used quite fits with algorithms based on trees, especially J48, REPTree and Random Forest. We took a significant number of samples (over 500 samples) and carried out different classification exercises using decision trees with different algorithms that allowed a quick and visual check of the quality of the features. That way we were able to guarantee, not formally (there are numerous techniques to validate this) but "visually", that the functionalities chosen were suitable for the initial scope of this research. Subsequently this has been validated in a more rigorous way.
For example, the first approaches revealed some characteristics that were very important decision-wise, such as the inclusion of the word "Document Close" (f43). However, this is not a definitive tree, since repeated iterations of the automatic selector are necessary to use all the features properly and enrich the final decision algorithm.
This exercise also aimed to debug the future classifier, the vector, and verify that decision trees could indeed result in a good classifier.

With these classification algorithms we also obtained the first approximations of the results that can be obtained with these features. In the specific case shown, we managed to classify a set of cross-validation test and training with an accuracy of 96.39%, and a false positive rate (misclassification) of 3.60%, which points to acceptable outcomes.

4.4 Structure of the Classifier and Basic Functioning

As we have seen in previous sections, the classifier receives as input a binary vector of features (predictors) previously selected by the ASC from the BV, which we will refer to as final vector (FV).
Since it is a binary classification problem, where all types with which we work are Malware and Goodware, as usual we have a training set the data of which are structured in the form $\{X_i, y_i\}_{i=1}^{Nv}$, where $y_i \in \{Malware, Goodware\}$ is the categorical variable and $X_i = [x_{i1}, \dots, x_{iN}] \mid x_i \in \{0, 1\}$ is each of the N_V vectors contained in the training set, formed by N binary predictors.
The characteristics that form the FV are those with a significant importance in the classification process according to the set of samples that exist in the system at a given time t. Thus, each x feature will have a p_i weighting at a given time. $p_i = P(x \mid t)$.
In turn, over time and depending on the samples incorporated to the system, different BV will appear among which we will need to select which one is most suitable depending on the time. Thus, at the time of writing this paper for example, the weight distribution in the BV of each feature according to its node's contribution in Random Forest is displayed in Fig. 4.
As we can see, from f40 the rest of the features are weighted 0, which does not mean that at another time it will remain the same.

4.5 Training and First Classification

Once we have a disposal of samples, a clear classification, and its characteristics vector extracted, we move to the construction phase of the

classifier itself. We have 1,671 samples classified according to the aforementioned criteria.

At the time of writing this paper, the Machine Learning algorithms built into the system are implementations of SVM, DT, RF and NN.

As a first approximation, we have chosen these four, as they are well known algorithms that have previously shown good results in similar classification problems. The implementation that has been used for all the aforementioned algorithms is provided by the scikit-learn framework for Python, which is widely known and used in these kind of problems.

For the test phase, we took the remaining 500 samples, where a 90% is goodware and a 10% is malware and we checked the results using the cross validation technique with a width of 10%.

5 RESULTS

5.1 Theoretical Validation of the Results

It is very important to note that we did not optimize the algorithms used, but we used a specific implementation amongst all existing variants that allow to work with these techniques.

On the other hand, we will not only take into account the final results in terms of accuracy, but for the evaluation results we will give special prominence to the confusion matrix, where "positive" means that a sample is malware.

This matrix should pay particular attention to false positives and false negatives in the case of analysis of malware samples. The reason is that, especially when compared with the malware world, not all accuracy is valid at any cost. Therefore, the construction of a classifier must also consider these parameters, ensuring that good results are not achieved at the expense of a false positives or false negatives rate that turn it inoperative.

In all cases, cross-validation was used at 10% and we calculated Accuracy, Precision and Recall. The results obtained in the test phase are displayed in Table 1. Additionally, we used F1-Score to combine accuracy and recall as the geometric average of both of them and AUC-ROC (Area Under ROC Curve).

During this test phase, different sets of training, tests and validation samples are used to build the classifier and tune it. Usually the precision achieved in classification has been high with all the algorithms. Neural networks reach a precision of 0.99%, makes it the most promising candidate for the later phase.

5.2 Practical Validation of the Results

The final stage checks the performance of the trained classifier with a series of samples never seen before. We have taken 267 new samples from which 55.8% (149) are considered malware according to the aforementioned criteria.

Table 1: Comparison of algorithms during the test phase.

Algorithm	Accuracy	F1-Score	Recall	Precision
Neural Networks	*0,99*	*0.961*	*0.925*	*0.99250*
SVM	*0,89*	*0.927*	*0.875*	*0.98045*
Decision Tree	*0,95*	*0.968*	*0.9375*	*0.99375*
Random Forest	*0.94*	*0.961*	*0.9375*	*0.99125*

From there, they are classified and their quality indices and the confusion matrix are compared according to the different algorithms used. The results are presented in Table 2.

Table 2: Comparison of algorithms during the final phase.

Algorithm	Accuracy	Precision	Precision media	AUC
Neural Networks	*0.90*	*0.86*	*0,91*	*0,95*
SVM	*0,93*	*0,93*	*0,94*	*0,95*
Decision Tree	*0,88*	*0,82*	*0,89*	*0,91*

Table 3: Comparison of the algorithms confusion matrix during the final test phase.

Algorithm	TP	FN	FP	TP
Neural Networks	*0.49*	*0.06*	*0.02*	*0.41*
SVM	*0.53*	*0.02*	*0,03*	*0,40*
Decision Tree	*0,46*	*0,08*	*0,02*	*0,41*

As the Table 2 shows, from the samples taken, we achieve a more precise classification with the SVM algorithm, since it allows to correctly classify 93% of the samples with a tolerable rate of false positives and false negatives. The AUC (Area Under the Curve ROC) can be used as a tool to measure the performance or effectiveness of the classifier. A test is considered as very good if it is between 0.9 and 0.97.

It is noteworthy how, with different samples (and contrary to what happened with the test set), the SVM classification algorithm appears as much effective than neural networks, with a higher precision and accuracy, which means that its performance is superior even taking into account false positives and negatives. To test this further, Table 3 details the confusion matrix for each of the algorithms. The highest rate of false negatives (up to 8%) occurs in the classification conducted by decision trees, in addition to having the poorer classification data in general. We can see very similar values between the neural networks and support vector machines, although again it is confirmed that the performance of support vector machines is especially remarkable. From the matrix we deduce that it is especially effective classifying true positives (malware that indeed is malware), and at the expense of false positives and negatives below 3%. We can see that the penalty of decision trees in Table 2 (with a precision of only 88%) is given by a false negative rate of 8%, that is, the rate of malware not detected.

6 CONCLUSION AND FUTURE WORK

The aim of this project is a first approach to the creation of a classifier with the capacity of learning to detect macro malware using mainly the characteristics of the VBA code, and compare its effectiveness between different algorithms and against traditional solutions such as antivirus engines. This kind of experiments does not seek to replace these consolidated traditional solutions, but to complement them and sometimes facilitate the work of analysts who design and update them.

During its development we have developed tools that allow the extraction and analysis of different types of documents, extracted and coded the features necessary for building a classifier, and finally we have compared the result of several classifiers previously trained. In addition, we have implemented the classifier in a framework that adds value to the results achieved by the classifier allowing us to improve, experiment and research further wih new data and algorithms.

However, and although the analysis and research still has room for improvement and optimization, we have to emphasize several points that have already been taken into account at the time of its implementation and development. For example, the fact that the use of antivirus engines as previous classification systems to train the algorithms makes the classifier inherit their successes, mistakes, advantages and disadvantages. To clarify this point, we develope some of the background to the analysis, in which it is presupposed that:

- Samples taken as goodware are actually goodware. Although risky, precautions taken when choosing these samples (such as not relying solely on antivirus engines, but on the source) guarantee "real" goodware to a larger extent than the simple classification by number of engines.
- Samples taken as malware by many antivirus, are actually malware.
- There is a strong time factor in detection: Engines need time to create signatures, and the freshests samples may go unnoticed until the specific signature is created and most engines start detecting it. The same happens with false positives: a not very detected sample may be detected because of a simple mistake that ends up being corrected by the engines. Thus, the detection threshold set to define a sample as malware or not can vary depending on the moment when that sample is taken and analyzed. Choosing a three-engines threshold, as has been the case, tries to adjust as closely as possible to the relation between early detection and a false positive.

In fact, for a comparison truly independent and disconnected from the classification already carried out by antivirus engines, we would need to properly validate the samples with a detailed manual analysis, which would eliminate the time factor. Regardless of these risk factors introduced and already mitigated as much as possible, during the research we have demonstrated that a first approach turns produces promising results, in which the best trained classifier works with precision above 90%, with a false positive and negative rate below 3%, making it a good filter comparable to the results of the most advanced antivirus, and demonstrating that the choice of characteristics intrinsic to the VBA code that forms a macro could become an effective method for the classification of malware.

REFERENCES

Chi, D., 2006. *Generic detection and elimination of*

marco viruses. United States of America, Patent No. US7089591 B1.

Ko, C. W., 2004. *Method and apparatus for detecting a macro computer virus using static analysis*. US, Patent No. US6697950 B1.

Lagadec, P., n.d. *Decalage*. [Online] Available at: https://www.decalage.info/python/oletools [Accessed 3 10 2016].

McAfee, n.d. [Online] Available at: https://www.google.com/patents/US6697950

Microsoft, n.d. *Microsoft*. [Online] Available at: https://support.office.com/en-us/article/Introduction-to-new-file-name-extensions-eca81dcb-5626-4e5b-8362-524d13ae4ec1? CorrelationId=bcd7dab6-5072-4b24-ab44-00819c4dabbe&ui=en-US&rs=en-US&ad=US&ocmsassetID=HA010006935 [Accessed 30 September 2016].

MMPC, 2015. *Microsoft TechNet*. [Online] Available at: https://blogs.technet.microsoft.com/mmpc/2015/04/27/social-engineering-tricks-open-the-door-to-macro-malware-attacks-how-can-we [Accessed 30 September 2016].

Nissim, N., Cohen, A. & Elovici, Y., 2015. *Boosting the Detection of Malicious Documents Using Designated Active Learning Methods*. s.l., IEEE.

Pornasdoro, A., 2014. *Microsoft*. [Online] Available at: https://blogs.technet.microsoft.com/mmpc/2014/12/30/before-you-enable-those macros/ [Accessed 30 September 2016].

Schreck, T., Berger, S. & Göbel, J., 2013. *BISSAM:Binary Instrumentation System for Secure Analysis of Malicious Documents*. Munich, Siemens CERT.

Shipp, A., 2009. *System for and method of detecting malware in macros and executable scripts*. US, Patent No. US7493658 B2.

Wikipedia, n.d. *Wikipedia*. [Online] Available at: https://en.wikipedia.org/wiki/Melissa_(computer_virus) [Accessed 30 September 2016].

Wikipedia, n.d. *Wikipedia*. [Online] Available at: https://en.wikipedia.org/wiki/Visual_Basic_for_Applications [Accessed 30 September 2016].