Approaches to Analysing Malware Received from a Reactive Network Telescope

Henning Krause
HAW Hamburg
henning.krause@haw-hamburg.de

ABSTRACT

Malware is a critical threat for the security in the Internet. Malicious actors and security specialists are constantly trying develop new ways to outperform one another. The goal of the criminals is to infiltrate a target system without being noticed. In order to protect a system it is an important step to identify malware before it is executed and start to invoke undesired behaviour.

With a reactive network telescope it is possible to obtain malware that is distributed by two-phase scanners. This offers the opportunity to collect malware at an early stage of distribution in which it may not be known by the security community yet. Therefore, a strategy will be developed to analyse and classify the findings. This work aims to review methods and practices for analysing malware in order to lay a foundation for the implementation of an analyses system to perform this task. Recent approaches, which try to identify malware as binaries or at execution time, will be presented.

KEYWORDS

Malware, Static Analysis, Dynamic Analysis

1 INTRODUCTION

Defining the term malware (*mal*icious soft*ware*) seems straight forward. Various definitions have been proposed which are very similar to each other and could be summarised as: *A software that has a harmful impact on other software without the user's content.* A harmful impact means that the malware changes the behaviour of the software in a way that it deviates from the intended behaviour. It causes incorrectness and damage as a consequence.

There are different kinds of malware. Well known are viruses, remote access Trojans (RAT, or Trojan Horse), spyware, worms, adware, scareware, bots, ransomware, cryptominers [15]. This list of types changes over time, as new use cases for malware appear. For example, with the rise of cryptocurrency since 2013, cryptominers became relevant. This is a classification based on the type. As Or-Meir et al. [15] pointed out that a classification based on the behaviour of the malware may be more precise, especially when an analysis is performed. They proposed the following classes: (I) stealing information, (II) creating a vulnerability, (III) denying service and (IV) executing commands from C&C. Classes are not exclusive since malware may serve several purposes.

These different purposes also lead to the question, what the motivations of the authors are to create and use it. A prominent example of malware is the Mirai botnet [1] which infiltrated IoT devices and abused them to perform massive DDoS attacks. But this is just one example which gained publicity. Kaspersky blocked 33.412.568 unique malicious objects in the year 2020 [10]. One might only speculate about all the different motivations. A chance to get an insight on the hacking community is offered by *CrimeBot*. It was created to crawl underground forums in order to allow research on a collection of posts. It collected 48m posts (ongoing) from 4 communities. Most of the data was obtained from the *hackforums*, which is a hacking community that became popular with the release of the Mirai botnet code, which was published by one of its users [16]. On these kind of forums illegal services such as malware as a service are offered . This indicates that the person who is realising the malware implementation and distribution may not be the stakeholder in the impact of the malware. All the posts that were collected over the time are stored in the *CrimeBB* which is available for researchers.

Bada and Pete [2] analysed the *CrimeBB*[16] dataset in order to explore the ecosystem around *Shodan*; a scan project for Internet facing devices and services which offers the user the possibility to search the results. Their work provides an insight on how and why users utilise the search engine. As *reconnaissance* is the first step of malware distribution their results offer an insight of the motivation and method of hackers. Bada and Pete [2] were able to identify several main use cases of *Shodan*. Building a botnet, which is is done by downloading and executing malware on vulnerable IoT devices, is of particular interest in the context of this project. Botnets may be used for several purposes such as DDoS attacks or cryptocurrenncy mining. In consideration of the fact that malware is easily accessible and underground forums provide various tutorials on how to find and get access to vulnerable devices, malware may be distributed by a wide range of people or organisations.

2 BACKGROUND

The malware which has to be analysed was captured by a reactive network telescope. Network telescopes are used to passively monitor traffic in the Internet. It usually observes a network prefix with provider-allocated IP addresses. The *UCSD Network Telescope*, operated by the *Center for Applied Internet Data Analysis*, is an example for a network telescope. A /8 network is monitored by this system [3]. Due to the fact that there are no active services in that prefix, the network telescope does barely receive legitimate traffic but rather Internet Background Radiation (IBR).IBR can be the result of a wide range of events such as address spoofing during a *DDoS* attack, misconfigurations (e.g., address mistypings), scanning activities or malware distribution [21]. A reactive network telescope elevates the functionality of a classic network telescope. It offers the possibility to engage into further interactions with the hosts of the network packets that arrive at the telescope.

The underlying work found a significant amount of irregular TCP SYN packets in the IBR. An irregularity indicates that they were handcrafted which gives one reason to suspect that those packets where send by stateless scanning tools. In this scenario the interaction functionality of the reactive network telescope becomes

particularly useful. A further investigation can be performed by replying to the irregular packets. Afterwards, the telescope received regular TCP SYN packets from the same host. Those could be used to establish a connection and receive payloads.

This behaviour may be referred to as two-phase-scanning. The first phase tests if a destination is online with light and fast scanning methods. If the first phase is positive, a second phase is initialised which includes further activities, such as port scanning, based on the goals of the scanner. This behaviour was also utilised by the Mirai Botnet [1]. It started with a rapid scanning phase and followed up with a brute force attack, if the the target was identified as a potential victim based on the gathered information.

2.1 The Problem of Malware Analysis

Malware authors are well aware of the fact that once the malware is being recognised, it will most likely be removed from the infected system. Furthermore, it is not in their interest that an analysis can be performed. Therefore, they spend a lot of effort on trying to avoid it. Loads of methods are utilised for that purpose.

Decompiling and analysing binaries is restrained by the efforts of the malware authors to make it as difficult as possible to reverse engineer the code. Transformations are applied that replace fragments of the code with semantically equivalent, but harder to analyse code. Even additional code may be added that does not change the behaviour of the program. This process is called *code obfuscation* [13]. Moser et al. [13] presented further obfuscating transformations that can be utilised. In order to analyse a program a control flow graph can be created which represents a possible flow of control by a sequence of instructions. *Control flow obfuscation* is the process of altering the flow in a way that makes it difficult to determine the next instruction in a graph. However, this process does not alter the control flow. This can be achieved by replacing unconditional jump and call instructions with another sequence. *Data Location Obfuscation* refers to the concept of hiding the actual data element that is accessed. The paper by Moser et al. [13] was published in 2007. It introduces general methods for obfuscation. As of today more complex and advanced methods exist, e.g., emulation-based obfuscation.

An overview about the challenges of dynamic analyses was given by Or-Meir et al. [15]. To evade an analysis the malware could behave differently when it detects that is is under analysis. Analysis tools and specially prepared systems leave indicators, which can be recognised by the malware. Indicators might be installed drivers, registry keys, user traces on the system and more. One more problem is that it is also possible that the malware will be running outside the scope of the analysis tool, e.g., on another privilege level.

To avoid the detection code obfuscation and network evasion are two of the main concerns. Code obfuscation is used to change the signature of the malware so that it will not be matching the known signature anymore and thus is not recognised. This makes code obfuscation a useful tool for malware authors that complicates static and dynamic analysis. Network evasion describes techniques that are used to hide the traffic to the command and control server.

In conclusion one can say that it can be very difficult to analyse malware, no matter which approach is chosen. However, due to the extensive expertise that is required to create such malware the chances for results are still realistic.

2.2 Related Surveys

The field of malware analysis is steadily changing and as a consequence lots of papers on this topic are published. The sheer amount makes it difficult to achieve a good coverage. Gandotra et al. [6] and Uppal et al. [18] published surveys in 2014 that focus on the difference between static and dynamic analysis. Or-Meir et al. [15] performed a survey on dynamic malware analysis methods in 2019. They claim that this is the first comprehensive survey on this topic since 2012. Recent focus has also be on machine learning techniques in order to improve existing methods. Ye et al. [24] published a survey that focuses on the data mining techniques. They review methods for feature extraction, classification and clustering.

3 STATIC ANALYSIS

This section introduces the static analyses of malware. First a general overview is given. Afterwards tools and frameworks are presented that approach different obstacles and steps during the static analysis. This provides a first idea of possible techniques and how they work together.

The static analysis of malware refers to the technique of analysing a malware sample without executing it [17]. The sample can be in form of a binary or source code. If the sample is a binary, the file has to be decompiled first. Disassemble tools like *IDA Pro* can be used for that purpose [24]. After the malware is decompiled, features can be extracted from the code. Using this features, detection patterns can be applied. For example the following patterns are of interest [24]:

System Calls Windows API calls are used by many programs and give a hint about the behaviour of the program. If they are considered altogether, higher semantics can be derived.
N-grams All substrings of the length *N*. Detection can be based on these strings.
Strings Sometimes the binaries contain interpretable strings that show the behaviour of the malware. If there is for example a string that includes a script in HTML format the desired output can be directly read.
Opcodes The machine language instructions can reflect the malware functionality.
Control Flow Graphs (CFGs) A graph that represents the control flow in a program.

Decompilation is a necessary step in the process of binary analysis. Human readable code is created from compiled programs. The problem is that even state-of-the-art decompilers cannot reproduce the original code and produce rather complex code which is difficult to understand. This process is further complicated by the affords of malware authors to prevent the decompilation. As explained in 2.1, code obfuscation, which drastically increases the difficulty of the decompilation process, is one of the main concerns in this matter.

Yadegari et al. [22] reported on a generic approach to automate deobfuscation of executable code. In particular they focus on reverse engineering *emulation-based obfuscation* and *return-oriented programming*. *Emulation-based obfuscation* obfuscates the code in a way that by examination only the emulation logic is revealed but

not the logic from the program itself. *Return-oriented programming* is a programming technique that results in complex control flows. The advantage of the generic approach is that it may be used for unseen obfuscation techniques, which is not possible with a specific technique. To keep the approach as generic as possible Yadegari et al. [22] aim to minimise the assumptions they make about the code; especially considering the type of obfuscation that was used. Their approach consists of the following steps:

Identifying Input and Output Values Input values are classified as values that are obtained from the command line, defined by a library routine, or read by an instruction of the program. Any value written by an instruction is defined as output.
Forward taint propagation After the inputs are identified all influenced instructions by the input should be found.
Code Simplification Semantics-preserving code simplifications are applied.
Control Flow Graph Construction A control flow graph is constructed which is used to apply semantics-preserving transformations to the control flow.

Yadegari et al. [22] evaluated their approach using a prototype implementation on obfuscated code. The result can be measured by using an algorithm to determine the distance between two control flow graphs. If two control flow graphs are similar the deobfuscation has been successful. To generate obfuscated files they used commercial emulation-obfuscation tools on several malware programs of different type. After obfuscation most of the programs showed a similarity of less than 10% compared to their originals. Using their approach they were able to generate a similarity of 85% after deobfuscation. This shows a significant improvement to former approaches. It should be noted that the results still vary a lot depending on the type of the program that is obfuscated. A total similarity could never be achieved.

As soon as the binary is deobfuscated further analysis can be applied. Yakdan et al. [23] developed the tool *DREAM*$^{++}$ which is a usability-optimised decompiler that focuses on increasing the readability of decompiler-created code. *DREAM* is a tool that uses the *IDA Pro* disassembler and builds the control-flow graph of functions in the binary. With *DREAM*$^{++}$ they try to improve the results of *DREAM*. During the evaluation they also considered *Hex-Rays* which is another decompiler similar to *DREAM*. They identified several problems that occurred with *DREAM*. Due to those problems the decompilation results in a code, that is difficult to understand even though the original code might not have been considerably complicated. An overview of the list of problems is shown in Figure 1. Complex expressions add further, unnecessary logic to the code which is hardly found in any code written by humans and thus are difficult to follow. Nested if statements and duplicate variables which represent the same value are an example for those. An example for convoluted control flow is the fact that binary code often contains duplicate code blocks as a result of macro expansion or other reasons. Thus, the decompiled code might contain the same block several times. An issue of the lack of high-lever semantic is for example the incapability of decompilers to recover variable names.

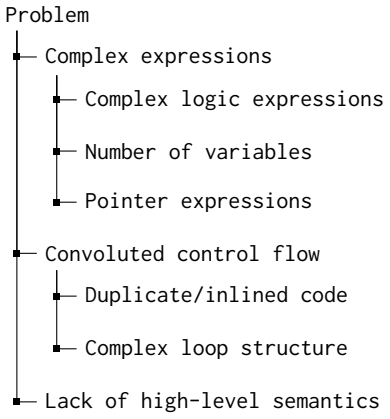

Figure 1: Yakdan et al. [23] identified problems for the DREAM decompiler. They organised their results in the shown three categories.

To improve the presented problems, Yakdan et al. [23] invented a set of optimisations that build upon the output of the *DREAM* decompiler. They developed three categories of semantics-preserving code transformations: (I) expression simplification, (II) control-flow simplification and (III) semantics-aware naming. Some of the improvements are: Duplicate variables are removed using congruence analysis, code is simplified by finding the simplest high-level form of logic expressions in the decompiled code, pointers are simplified, loops are transformed, functions are outlined and variables are renamed.

In a user study they showed that the improvements were a considerable help for the analysts. A code example shows that *DREAM*$^{++}$ produces smaller and simpler code. The result of Hex-Rays was 41 lines of code long, *DREAM* resulted in 27 lines and *DREAM*$^{++}$ managed to pin it down to 13 lines (see Yakdan et al. [23] for more detailed examples).

4 DYNAMIC ANALYSIS

This section starts with an overview based on Or-Meir et al. [15]. Afterwards a description of the frameworks by Duan et al. [5] and Mohaisen et al. [12] will be given in order to create an understanding of possible implementations.

A dynamic analysis refers to the technique of observing the behaviour of a software during execution. The goal is to identify the malicious actions which are execute. At the same time the analysis platform must be protected from being compromised. The main idea of a dynamic analysis is to prepare a framework composed of three components: (I) malware sample, (II) hardware and operating system and (III) the analysis tool. It is important that the system is clean before starting an analysis in order to avoid side effects from previous actions or configurations. The malware could be run directly on a pc, in a virtual machine, on a *type 1 hypervisor* or on a full system emulation. There is a also side-channel data acquisition which refers to a technique to analyse an electronic device by monitoring the physical power consumption. Different analysis techniques are the following [15]:

Function Call Analysis Examining which functions a malware is calling in order to derive its behaviour by the bulk of calls.
Execution Control Checking the state of the malware and the OS by utilising techniques such as debugging.
Flow Tracking Following the information through the code that is executed by the malware.
Tracing Executing code leaves information behind. This information can be gathered and analysed.
Side-channel Analysis Analyse the behaviour of physical components by their power consumption, electromagnetic emissions or internal CPU events.

The first framework we look at is *Detective* - a tool to identify and analyse malware processes in Windows. It was published by Duan et al. [5]. They address the issue that during the analysis of processes there may be a considerable amount of processes that are unknown to the investigator. *Detective* solves that problem by automatically classifying benign and malware processes. Furthermore it is capable of describing malware behaviour on a high semantic level and also "shedding light on evidence collection"[5].

Their idea is based on the observation that processes can be classified by the set of Dynamic-Link Libraries (DLLs) they load. The Windows API is implemented as a set of DLLs which can be used to derive potential capabilities of a process. Although Detective is currently designed for the Windows operating system, the core concept which is utilised can also be applied to other operating systems. Similar processes most likely load a similar set of DLLs. By clustering the analysed processes by the DLL set it is possible to predict the behaviour of an unknown process. This way it is possible to identify an unknown malware process if it shows comparable functionality, as it is indicated by the DLLs, to previously known processes. They evaluated several clustering algorithms and finally chose DBSCAN. One of the main reasons was that it does not require a specification of the number of clusters. In order to achieve the classification they created a set of key DLLs which is a subset of all possible DLLs. This way they reduced the dimension of the model. To train the model a training data set was created with both benign and malware processes included. The malware processes for this purpose were obtained from VirusTotal[1]. An advantage of using a training data set by VirusTotal is that the data usually contains tags that where given by antivirus vendors. These tags are later used to describe the behaviour of the process. On top of that each DLL has a specific purpose and if they are observed in a group they indicate the behaviour of the malware.

Duan et al. [5] evaluated the performance by using a training, validation and two test data sets. The test dataset included 20/25 malware DLLs with a total of 24/30 DLLs. *Detective* achieved a correctness of 92% on the task of identifying malware with two false positives for each test dataset.

Another framework was proposed by Mohaisen et al. [12]. They introduced *AMAL* - "an operational and large-scale behaviour-based solution for malware analysis and classification" [12]. *AMAL* is a combination of two sub-systems.

The first subsystem is *AutoMal* which analyses the behaviour based on memory (e.g., volatile memory), the file system (e.g., created/deleted files), registry (e.g., created/deleted/modified registry content) and network (e.g., DNS resolution, exchanged data). With those information a profile for a malware is generated. Furthermore, behaviours are summarised into artefacts. It is based on VMware, supports a variety of configurations and is very flexible. For example, various types of malware samples are allowed, such as EXE, DLL, PDF, etc., the OS may be reset and/or configured before each sample and the privileges on the OS can be configured. Its system architecture consists of several components which allows *AutoMal* to run multiple samples at the same time. The components and their functionality can be summarised as follows (see [12] for a more detailed description):

Sample submitter Responsible for providing samples to the *AutoMal*. Multiple sources and priorities are supported.
Controller The controller takes care of the test setup which includes choosing a sample and configuring the VM. The controller also performs the artefact collection.
Workers An individual VM where an analysis will be performed. They are functionally independent of the controller.
Database A MySQL database storing all the results.

MaLabel is the second subsystem. It is a collection of multiple techniques and algorithms for clustering and classification based on the low-level artefacts provided by *AutoMal*. The representative features of the collected data is extracted and used for behaviour-based grouping and labelling of malware. *MaLabel* offers an evaluation of the performance of the implemented algorithms while leaving the final decision which to use to the user. Even though *AutoMal* provides more features, which could be used for classification and clustering, *MaLabel* only uses a subset of the following groups of features: file system, registry and network features.

Mohaisen et al. [12] evaluated *AMAL* performance against a small dataset and against a large dataset containing 115,157 samples. In the bigger dataset the share of malware samples reduced in order to imitate a real world use case. Using the smaller sample they achieved 99% precision and recall. The larger dataset resulted in a performance of around 90%. For the false positives they named the following reasons: imperfect labels created by census over antivirus scans and not analysts, errors resulting from the smaller share of malware samples in the training data and results attributed to different malware families with a similar context.

Wang et al. [19] claim that there are some limitations to the dynamic analysis of malware. The use of a specialised environment for the execution of the malware in order to monitor its behaviour may be recognised by the malware. For example the malware could present inconspicuous behaviour if executed in a virtual environment. Furthermore, certain kinds of malware, e.g., Trojan, do not present noticeable behaviour until notified. A further drawback is the fact that a malware may only target a certain OS or certain hardware.

5 FURTHER APPROACHES

In this section approaches to malware analysis are presented that do not fit in either the static or the dynamic category, but are still interesting and should be mentioned. They have in common that their aim is to identify features of the malware and use those to detect new malware.

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

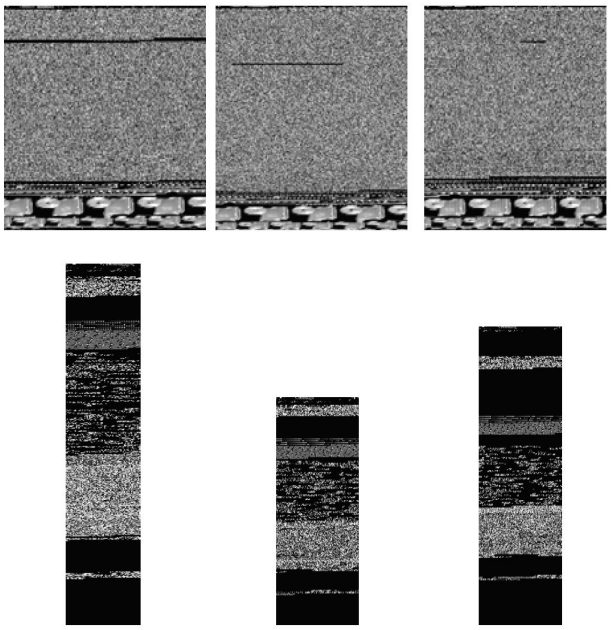

Figure 2: Images of visualised malware. The first row shows three instances of malware of the Fakerean family and the second row shows three instances of the Dontovo family. Published by Nataraj et al. [14] in their paper.

5.1 Malware Visualisation

A novel approach at that time (2011) was published by Nataraj et al. [14]. They used image processing techniques to visualise and classify malware. Neither disassembly nor code execution is necessary to use this method. However, one could argue that there is a greater similarity to static analysis. This can be argued by the fact that one of the main characteristics of the dynamic analysis is the required execution of the sample.

The approach take advantage of the fact that a binary can be presented as a string of zeros and ones. When the malware is read as a vector of 8 bit unsigned integers, it reorganised into a 2D array and visualised as a gray scale image. Examples of those visualisations can be seen in Figure 2. It clearly stands out that malware of the same family looks similar to each other. The length of the image is given by the size of the malware file. This not only makes it possible to see the visual similarities of different malware samples from the same family, but also allows a further analysis. Feature vector and classifier can be applied on those images. The texture analysis by the frequency of a texture block is an example for those methods. For the implementation in the paper they used GIST in combination with k-nearest neighbours for classification.

When evaluating their approach against a large dataset of 9458 malware with 25 different families they achieved an accuracy of 98%. It should be noted that they state that their approach can be limited by counter measurements such as relocating sections or adding redundant data in the binary.

A similar approach was proposed by Gibert et al. [7]. They use the same technique to represent a malware file as a gray scale image. However, they set the file size to a fixed size. This is mandatory step since they are using a Convolutional Neural Network (CNN) to group the malware files. Their technique improves the computational performance and accuracy of previous, similar approaches.

5.2 Entropy Analysis

Entropy approaches to analysing files have been used for several years. The first automatic identification approach was proposed by Lyda and Hamrock [11], who created a tool for automatically identifying encrypted or packed malware executables. Prior to their work, other approaches were not automated [20] or used entropy for other approaches, e.g., identifying network-based attacks [8].

The work of Jochheim et al. [9] aimed to identify embedded malicious code using entropy and signal processing techniques. In order to apply the identification, several steps have to be performed. First, an entropy-function is build which indicates different data types and code sections within file. Afterwards a frequency analysis is applied to that function. Finally, the result is applied on an Artificial Neural Network (ANN). This allows the identification of binary-instruction code. They showed their approach has a good performance in test and real-world conditions.

Cox et al. [4] proposed another concept for malware analysis. They preprocessed the data into fixed-length representations in order to allow the usage of a feedforward neural network as the classifier. Three different methods were used to calculate probabilities: (I) Shannon entropy within a sliding window, (II) byte distribution and (III) power spectral density. The entropy approach showed the worst performance with a 77.11% accuracy which Cox et al. [4] explained the fact that PNG and GIF files were often confused with AES-256 encrypted files. By combining all three methods they were able to achieve a 97.44% accuracy.

6 CONCLUSION AND FUTURE WORK

In this work, we gave an introduction to the analysis of malware. The difficulties that occur during malware analysis were presented and it was shown that various ways for malware authors exist to make it even more challenging. Some of the many existing approaches to perform an analysis were presented. The goal was to give the reader a basic understand of the wide variety of techniques that exist.

In conclusion, it can be said that each approach has its own strengths and weaknesses. While some rely on a training data set and perform the classification automatically afterwards, others need manual investigation but offer a good framework to perform. The goal of the analysis should be considered. If only a classification is of interest, a dynamic approach may be the right choice. If the specific behaviour of a file has to be investigated, a static analysis would give promising results. Considering the case of the malware files from our responsive network telescope a good starting point could be the static analysis which does not have the overhead of the dynamic analysis with the need to setup a safe execution environment.

This work created a basic understanding of methods that are used and therefore lay a foundation for future work. As the tremendous

amount of publications exceeded the scope of this work, a further survey on recent publications is of interest. The future research can be focused on the topic that is the most promising for this use case. Once decided upon a tool or framework an implementation and evaluation of it should be realised. As both static and dynamic analysis techniques have their advantages and disadvantages a hybrid approach could also be considered in which the best of both techniques are combined.

Once the malware was analysed the results should be carefully documented. Furthermore, a classification could help to find patterns in the distribution. Among the possibilities to classify the malware, a classification by type or by behaviour could be chosen.

REFERENCES

[1] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium* (Vancouver, BC, Canada) *(SEC'17)*. USENIX Association, USA, 1093–1110.

[2] Maria Bada and Ildiko Pete. 2020. An exploration of the cybercrime ecosystem around Shodan. In *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)* (Paris, France). IEEE International Conference on Internet of Things: Systems, Management and Security, Paris, France, 8. http://emergingtechnet.org/Procs/IOTSMS2020/22/RC_IOTSMS2020_22.pdf

[3] CAIDA. 2021. *The UCSD Network Telescope.* CAIDA: Center for Applied Internet Data Analysis. https://www.caida.org/projects/network_telescope/

[4] Jonathan A. Cox, Conrad D. James, and James B. Aimone. 2015. A Signal Processing Approach for Cyber Data Classification with Deep Neural Networks. *Procedia Computer Science* 61 (2015), 349–354. https://doi.org/10.1016/j.procs.2015.09.156 Complex Adaptive Systems San Jose, CA November 2-4, 2015.

[5] Y. Duan, X. Fu, B. Luo, Z. Wang, J. Shi, and X. Du. 2015. Detective: Automatically identify and analyze malware processes in forensic scenarios via DLLs. In *2015 IEEE International Conference on Communications (ICC)*. IEEE Computer Society, London, UK, 5691–5696. https://doi.org/10.1109/ICC.2015.7249229

[6] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. 2014. Malware Analysis and Classification: A Survey. *Journal of Information Security* 05 (01 2014), 56–64. https://doi.org/10.4236/jis.2014.52006

[7] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. 2019. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques* 15, 1 (01 Mar 2019), 15–28. https://doi.org/10.1007/s11416-018-0323-0

[8] Jean Goubault-Larrecq and Julien Olivain. 2006. *Detecting Subverted Cryptographic Protocols by Entropy Checking.* Technical Report LSV-06-13. Laboratoire Spécification et Vérification.

[9] Benjamin Jochheim, Thomas C. Schmidt, and Matthias Wählisch. 2011. A Signature-free approach to malicious code detection by applying entropy analysis to network streams. In *Proc. of the TERENA Networking Conference* (Prague, Czech Rep.). Terena, Amsterdam, Poster. https://tnc2011.terena.org/core/poster/29

[10] Kaspersky. 2020. Kaspersky Security Bulletin 2020. https://go.kaspersky.com/rs/802-IJN-240/images/KSB_statistics_2020_en.pdf

[11] Robert Lyda and James Hamrock. 2007. Using Entropy Analysis to Find Encrypted and Packed Malware. *IEEE Security and Privacy* 5, 2 (2007), 40–45.

[12] Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen. 2015. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security* 52 (2015), 251–266. https://doi.org/10.1016/j.cose.2015.04.001

[13] A. Moser, C. Kruegel, and E. Kirda. 2007. Limits of Static Analysis for Malware Detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, FL, USA, 421–430. https://doi.org/10.1109/ACSAC.2007.21

[14] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security* (Pittsburgh, Pennsylvania, USA) *(VizSec '11)*. Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/2016904.2016908

[15] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. 2019. Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Comput. Surv.* 52, 5, Article 88 (Sept. 2019), 48 pages. https://doi.org/10.1145/3329786

[16] Sergio Pastrana, Daniel R. Thomas, Alice Hutchings, and Richard Clayton. 2018. CrimeBB: Enabling Cybercrime Research on Underground Forums at Scale. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1845–1854. https://doi.org/10.1145/3178876.3186178

[17] Rami Sihwail, Khairuddin Omar, and Khairul Akram Zainol Ariffin. 2018. A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. *International Journal on Advanced Science Engineering Information Technology* 8 (09 2018), 1662. https://doi.org/10.18517/ijaseit.8.4-2.6827

[18] D. Uppal, Vishakha Mehra, and V. Verma. 2014. Basic survey on Malware Analysis, Tools and Techniques. *International Journal of Computer Science & Applications* 4 (2014), 103–112.

[19] C. Wang, Z. Qin, J. Zhang, and H. Yin. 2016. A malware variants detection methodology with an opcode based feature method and a fast density based clustering algorithm. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE Computer Society, Changsha, China, 481–487. https://doi.org/10.1109/FSKD.2016.7603221

[20] M. Weber, M. Schmid, M. Schatz, and D. Geyer. 2002. A toolkit for detecting and analyzing malicious software. In *18th Annual Computer Security Applications Conference, 2002. Proceedings.* IEEE, Las Vegas, NV, USA, 423–431. https://doi.org/10.1109/CSAC.2002.1176314

[21] Eric Wustrow, Manish Karir, Michael Bailey, Farnam Jahanian, and Geoff Huston. 2010. Internet Background Radiation Revisited. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (Melbourne, Australia) *(IMC '10)*. Association for Computing Machinery, New York, NY, USA, 62–74. https://doi.org/10.1145/1879141.1879149

[22] B. Yadegari, B. Johannesmeyer, B. Whitely, and S. Debray. 2015. A Generic Approach to Automatic Deobfuscation of Executable Code. In *2015 IEEE Symposium on Security and Privacy*. IEEE, San Jose, CA, USA, 674–691. https://doi.org/10.1109/SP.2015.47

[23] K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith. 2016. Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, San Jose, CA, USA, 158–177. https://doi.org/10.1109/SP.2016.18

[24] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* 50, 3, Article 41 (June 2017), 40 pages. https://doi.org/10.1145/3073559