Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-023-00476-z

ORIGINAL PAPER

Recognition of tor malware and onion services

Jesper Bergman[1] · Oliver B. Popov[1]

Received: 11 October 2022 / Accepted: 16 March 2023
© The Author(s) 2023

Abstract
The transformation of the contemporary societies through digital technologies has had a profound effect on all human activities including those that are in the realm of illegal, unlawful, and criminal deeds. Moreover, the affordances provided by the anonymity creating techniques such as the Tor protocol which are beneficial for preserving civil liberties, appear to be highly profitable for various types of miscreants whose crimes range from human trafficking, arms trading, and child pornography to selling controlled substances and racketeering. The Tor similar technologies are the foundation of a vast, often mysterious, sometimes anecdotal, and occasionally dangerous space termed as the Dark Web. Using the features that make the Internet a uniquely generative knowledge agglomeration, with no borders, and permeating different jurisdictions, the Dark Web is a source of perpetual challenges for both national and international law enforcement agencies. The anonymity granted to the wrong people increases the complexity and the cost of identifying both the crimes and the criminals, which is often exacerbated with lack of proper human resources. Technologies such as machine learning and artificial intelligence come to the rescue through automation, intensive data harvesting, and analysis built into various types of web crawlers to explore and identify dark markets and the people behind them. It is essential for an effective and efficient crawling to have a pool of dark sites or onion URLs. The research study presents a way to build a crawling mechanism by extracting onion URLs from malicious executables by running them in a sandbox environment and then analysing the log file using machine learning algorithms. By discerning between the malware that uses the Tor network and the one that does not, we were able to classify the Tor using malware with an accuracy rate of 91% with a logistic regression algorithm. The initial results suggest that it is possible to use this machine learning approach to diagnose new malicious servers on the Tor network. Embedding this kind of mechanism into the crawler may also induce predictability, and thus efficiency in recognising dark market activities, and consequently, their closure.

Keywords Tor · Malware · Machine learning · Forensics

1 Introduction

The generative nature of digital technologies has transformed parts of the society from kinetic or analogue into non-kinetic or digital, creating a hybrid social, economic, and cultural space termed as Cyber Physical Systems (CPS). Digital transformation, among others, has had a profound effect on all human activities including those that are in the realm of illegal and criminal deeds. For instance, the Tor protocol, almost a synonym for Anonymous Communication Networks (ACNs), affords through anonymity-granting techniques preservation of civil liberties. Nevertheless, the same protocol appears to be highly profitable for miscreants whose crimes range from human trafficking, arms trading, and child pornography to selling controlled substances and racketeering.

Tor similar technologies are the foundation of vast and occasionally dangerous space termed as the Dark Web. With no borders, and permeating different jurisdictions, the dark web is a source of perpetual challenges for national and international law enforcement agencies. The anonymity increases the complexity and the cost of identifying both crimes and criminals, which is often exacerbated with a lack of proper human resources [15]. However, digital technologies have also created a multitude of techniques and tools, for instance machine learning, artificial intelligence, intensive data har-

✉ Jesper Bergman
jesperbe@dsv.su.se

Oliver B. Popov
popov@dsv.su.se

[1] Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden

Published online: 29 April 2023

vesting, and analysis, that can be automated and built into web crawlers to explore and identify dark markets and the people behind them. It is an imperative for an effective and an efficient crawling to have a pool of dark sites or onion URLs.

The research study shows the transformed digital crime landscape. It also provides some insight into building "intelligent" crawlers by extracting onion URLs from malicious executables. The enumeration of a few results concerning malware identification and seizure on the Dark web starts with the work by [51] who use text analysis to attribute child sexual abuse material posted on the dark web. Similarly, Chen et al. [6] collected and analysed almost ten thousand web pages related to jihad terrorism activities on the world wide web (or the "clear web"). Frank [18] report on a collection of information from the dark web and the use of data analysis algorithms to discern between extremist and non-extremist content.

Kwon et al. [29] studied the dark web forum content spanning over more than two years, assessing the nature of the communication between the forum users. Nunes et al. [35] classified products from dark marketplaces that even made it possible to predict previously unseen items in an automated way.

In contrast, Westlake et al. [60] question the validity of child sexual abuse material analysed automatically by web crawling techniques. Namely, the risk of failing to select relevant criteria could lead to a distorted description of the quality of the analysed data [60, p. 703].

The analysis of web content and Tor onion services is only possible when the address to the corresponding server is known. Therefore identifying new onion services partaking in cyber criminal activities is necessary. Due to the anonymity, identifying and monitoring cyber criminal hidden services is a demanding and challenging endeavour [15].

The use of malicious software to anonymously communicate with the Tor command and control hidden services and to host ransomware payment instruction websites is well documented [7]. Indeed, malware in general, and ransomware in particular, appear to be rather profitable for cybercriminals. A timeline of ransomware presented by [16] shows that while ransomware is somewhat decreasing in popularity, it is still a threat and has increased the overall monetary value of the extortion [16].

Several researchers such as Pirscoveanu et al. [39] have previously explored the possibilities of using machine learning classification algorithms to automatically categorise malware based on the behaviour of the malicious software (application programming interface (API) calls, registry key modifications, etcetera). Pirscoveanu et al. [39] categorised malware such as trojan, adware, potentially unwanted programs (PUP), and worm with a precision score of over 90% for each respective category.

By using similar machine learning algorithms, malware could be classified based on its dependency on the Tor network or not. Hence, in the paper, Tor-related classes of miscreant services and corresponding servers that can be monitored and further investigated were identified. We also discuss how this technique can be used in practice to aid law enforcement agencies.

The disposition of the paper is comprised of six sections and a bibliography. The first section termed as Introduction (this one) describes briefly the nature of anonymous communication networks, malware analysis, and the challenges of identifying cybercriminal services on the Tor network.

In the second section Related work, a more in-depth presentation of previous research is given together with a detailed explanation of malware analysis, machine learning, web crawling, and the Tor network. The research methodology is presented in Sect. 3, and it employs an empirical study using an experiment based strategy. Section 4 provides all the research results from the experiments, while the related discussion is placed in Sect. 5. Finally, Sect. 6 presents the conclusions and the plausible directions for future work.

2 Related work

The section outlines some of the basic ideas and concepts behind anonymous communication networks (ACNs), in particular the Tor network, and the concept of crawlers and their varieties on Tor. In addition, the authors discuss malware and the possibilities for its automated analysis, classification, and the specifics of its presence on the Tor network. Some of the motivational aspects behind the research are also addressed.

2.1 The tor network

The Tor network consists of "onion routers" (OR) - servers that act as traffic relay nodes through the network. We recognise three different types of relays: entry, exit, and middle ones. When connecting to the Tor network, these relays form virtual circuits (or paths), through which the traffic is routed. The entry nodes designated as trusted (which entails they are stable and reliable for connectivity and conformity to the protocol) initiate the circuit, the middle relays (at least one for each triple) extend the circuit, and dedicated exit relays provide a connection to a destination server (Fig. 1).

The encryption, which is the sine qua non of all anonymous networks including Tor, is applied to the respective network traffic in layers that resemble the shape of an onion. This is where the term "onion routing" originates from. While the minimum number of nodes needed to establish a circuit is three, it is possible to have more than one middle relay in a circuit. Each node knows only the node it is sending the

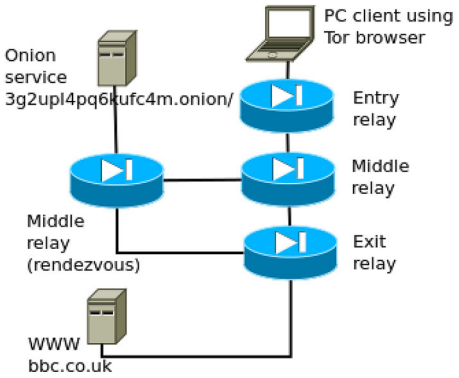

Fig. 1 Simplified overview of the Tor network's components

traffic to and the node it is receiving the traffic from. Thus, there is no a single node that knows the entire structure [13].

In addition to the ORs, there are nine central Directory Authority (DA) servers controlled by a number of selected trusted parties around the world [57]. The DA servers share information about the nodes in the network that is cached by other ORs as a Directory cache. The directory cache contains information about the nodes in the network, such as up-time, donated bandwidth, consensus weight (which is a qualification of their trustworthiness and reliability), and the identity key belonging to the OR [56].

The DA servers are the only central points that come hard-coded with the installation package of the Tor software. The Directory servers also furnish the necessary descriptors for reaching onion services or the servers only reachable within the Tor network [56].

2.1.1 Tor onion services

Onion services are simply servers on the Tor network that uphold the Tor principles to establish a "non-traceable" route to the host. An onion service has several trusted "introductory points", ORs that could also work as entry and middle relay nodes. Within the network they can be used to connect to a service via a "rendezvous point" (RP) which is an OR as well. The introductory points direct the requester (or the one that solicits a service) to the rendezvous point. Then the client and the onion service communicate via the RP which prevents the service from revealing its location. [58].

2.1.2 The.onion top level domain

An onion service on the Tor network has a public key that is used to derive its associated the URL. In the Tor protocol version 2 (v2), the derivation was done by taking the 16 first characters of the SHA1 of the RSA key of the onion service and encode it using base32, and then adding".onion" as the top-level domain.

As of version 3 (v3), the domain name is comprised of the base32 of the public key (an ED25519 key), its checksum, and the version number (1 byte) [58]. A v2 of an onion service's URL would look similar to:
"yyhws9optuwiwsns.onion" whereas a v3 server would have a URL akin to:
"l5satjgud6gucryazcyvyvhuxhr74u6ygigiuyixe3a6ysis67ororad.onion"
Reaching an onion service assumes knowledge of the URL to it. There are services on the Tor network that index known URLs to onion services and offer search engine services for the visitors (for example https://ahmia.fi). Nevertheless, there is a slight difference between the ways it works on the Tor web compared to the clear web. The Tor network does not use IP addresses for reaching onion services; it only refers to URLs with the top level domain .onion. Therefore, onion services cannot be identified by scanning the closed IPv4/IPv6 space, which is possible on the IP based Internet. The identification of the servers on the Tor network is pertinent to the the capability of the search engines to obtain the URLs from somewhere. Hence, the corroboration of new onion services becomes a resource intensive task. Using a brute force to "guess" an onion URL for a v3 onion site would mean 56 characters with a 32 character key-space (A-Z and 3-7), i.e. the number of possible combinations would be 56^{32}, which is equal to 875017752602483387956491386392423776294522678519644815 36.

Furthermore, for any URLs guessing it would be necessary to send a request to check their availability on the Tor network. While technically possible, it is not a very efficient way of doing it. This is a privacy maintaining and preserving feature of the Tor protocol, though when used for criminal purposes it turns into a problem for law enforcement agencies (LEAs).

2.2 Crawling onion services

After the infamous seizure and investigation of the Silk road in 2013, much of the research in the interplay between cybercrime and ACNs, has focused on analysing marketplaces similar to the Silk Road such as AlphaBay [53], Hell [41], Darkode [41], Black Market Reloaded [51, 53], Agora, and Valhalla [53].

One technique for collecting evidence from websites on ACNs is to collect and store their content automatically as historical copies. The archived data can be analysed later and eventually included in an investigation. The automated collection and archiving of websites is often referred to as "crawling". Crawling is merely a simple paraphrase for automated collection of web content.

By replicating the web crawling techniques applied by search engine companies such as Google, Yandex, and Baidu,

dark marketplaces can be indexed and analysed. They are usually protected by authentication pages, yet reachable and accessible via the right automation techniques.

The obvious problem of finding URLs when crawling ACNs is aggravated with the fact that there are a very few onion sites that link to each other. Usually, administrators of illegal websites tend to keep their exposure to a minimum and to avoid linking to competing sites. As a result of this, visual graphs of Tor Onion websites and their links have shown that a small number of servers have a large number of outgoing links, these are the link collections and search engines and the rest have few outgoing links [4]. There are ACN crawlers available on the net that collect onion URLs from different websites on the world wide web WWW and archive the web content as a digital copy, e.g. [8, 26].

Using historical copies of ACN websites as part of an investigation and possibly also as evidence in a court of law requires admissible and forensically sound evidence, as in the case with the dark marketplaces Silk Road 2.0 [12] or AlphaBay [43].

In addition to scraping web content from web servers on ACNs like Tor, information about the servers can be scraped as well; a command and control server for a botnet can, for example, be scanned and monitored to see possible META data such as operating system and other software versions, uptime, open ports, and more. In addition, law enforcement might be obliged to attack an Onion Service in order to disrupt it, given that the.onion URL is know [46].

Popov et al. [40] suggest a conceptual model of an intelligent dark web crawler that emphasises the overarching goal of the artefact forensic soundness.

In the current article, by using machine learning based classification, we present a way to identify Tor dependant malicious software. The proposed approach enables us to extract onion site URLs that can be used later by the crawler to (1) work effectively and efficiently by finding new Onion Services, and (2) produce a better training data for improving prediction and classification as the design of the crawler presented in Popov et al. [40].

2.3 Malware analysis

Malware is generally executed in an operating system (OS), since it needs to communicate with the built-in components which include, among other things, the software application programmable interfaces (API) and libraries. The components are also manipulated by the malware to communicate with a command and control servers that send instructions to the malware on an infected computer. API calls and libraries are essential in malware analysis since they reveal behavioural patterns.

Windows is the largest operating system amongst desktop computers [52] which implies that most of the malicious software is written for the Microsoft OS. Windows based software, either malicious or benign, uses several different components to interact with the hardware and the operating system. Some of these components are Dynamic Link Libraries (DLLs), API calls, registry keys, mutexes, and atoms [30, 31]. These components can provide malware analysts with valuable clues relative to the operation of the examined programs.

DLLs are exercised by Windows software to make system calls to the file system, parts of the hardware, and the registry [30]. Therefore, the DLLs can provide an essential insight during the malware analysis process.

Mutexes, or mutex objects, are mutually exclusive objects owned by only one process at a time which makes them suitable for process synchronisation. There is a history of using mutexes by malware authors in marking an infected system to avoid repetitive infections of the same systems. [31, p. 434].

2.4 Automated malware analysis and classification

Analysing malware is often a time consuming and tedious task. It is done either by manually inspecting the machine code of the malware executable or (semi)automatically by running the malware in an isolated environment (a *sandbox*) to assess the behaviour of the components. It is a common approach to analyse a number of malware samples by employing a set of different methods such as association rules, support vector machines, decision trees, random forest, and naïve Bayes algorithms [19].

The extensive literature review by [19] posits that machine learning based malware classification is useful to combat the increasingly sophisticated obfuscation techniques for malware. Although efficient and effective, the authors argue that large quantities of collected malware behaviour could result in imbalanced data sets that might skew the classifications.

Meng et al. [34] second that malware data sets might be imbalanced due to few benign samples in relation to malicious samples, and also amongst different malware families the proportionality might be skewed due to a higher prevalence of one family and not the other.

Alazab et al. [2] experimented using static analysis of Windows executables consisting of (1) disassembly of the malware (2) extraction of API calls, and (3) mapping of API calls with Microsoft's software developments documentation pages. While not referring to any machine learning algorithms, Alazab et al. [2] still underline the value of API call analysis of Windows executable files to categorise them as either benign or malicious.

Pirscoveanu et al. [39] employed Cuckoo Sandbox[1] to classify malware in different categories depending on their

[1] https://cuckoo.org.

behaviour. They used API calls of Windows executables to train a support vector machine (SVM) and classify malware executables into their respective categories: Trojan, adware, rootkit, and potentially unwanted program (PUP) Pirscoveanu et al. [39].

Tian et al. [55] did a similar classification study based on static analyses of ASCII strings present in the executables which included strings, API calls, registry keys, and other clear text data that was available in disassembly tool IDA Pro. The authors achieved an accuracy rate of over 97% using a random forest classifier based on the executable strings [55].

Shalaginov et al. [49] applied a machine learning approach based on static analyses of portable executable (PE) headers of malicious and benign Windows software. By using information gain and Correlation-based Feature Subset Selection (CFS) algorithms for feature selection, the researchers successfully classified malware using C4.5 and k-NN with an accuracy of circa 97% at best.

Pektas and Acarman [38] explored ensemble learning with a random forest algorithm to classify Android malware with an accuracy level of 92%. They created a data set for training classification models by using behavioural analyses from Cuckoo sandbox. In addition to the random forest algorithm, the authors also reached for linear regression and naive Bayes classification with accuracy rates of 91% and 83% respectively.

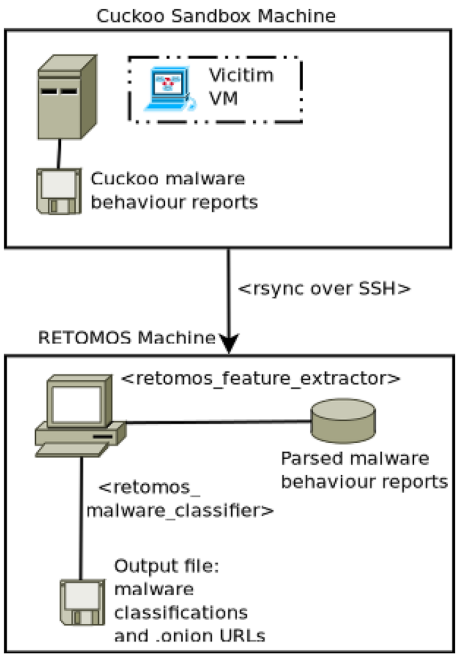

Fig. 2 Topology of the research experiment setup

2.5 Tor facilitated malware

Ling et al. [32] developed "TorWard"-a tool for discovering, blocking, and tracing malicious traffic on the Tor network. The anonymity afforded by the Tor network leaves a few digital footprints when the malware communicates with the operational servers.

The same feature was behind the incognito malware communication in the case of "CryptoWall" mainly based on Tor with occasional overtures to I2P protocol to connect to its command and control servers [50]. Similarly, Tor benefits were extended to the work of "Onion Ransomware" [28], "ZeuS" [54], Retefe [23, 61], and REvil [36] for their malicious operations. Another example is "Eleanor" - a Mac OS X malware that utilised the Tor network to set up an onion service on the infected system, allowing the operators to connect to the victim anonymously [21, 44].

A Tor based malware detection system presented by [20] introduced lists of IP addresses to entry relays as a detection mechanism for Tor facilitating software - both benign and malicious. Complementary sources such as the correlation of IP traffic to known domain flux botnets helped in discerning as malicious a host connectivity in Tor and flux botnets [20].

According to [5], there are both advantages and disadvantages of facilitating the Tor network for communicating malware.

The authors concluded that (1) Tor is used by malicious software to receive operational commands and to share and distribute data, and (2) that the Tor protocol creates obstacles to the anonymous communication channel due to the preference of domain names over IP addresses. Namely, the command and control servers cannot hide multiple IP addresses behind one single domain on Tor [5].

3 Research method

In this research study, we used an experiment based, quantitative research method to address the research problem, since proper analytics of large quantities of data requires efficient and effective machine learning applications. The experiment was based on data collected from a malware behaviour analysis system, which was later transformed into a data set and subjected to machine learning algorithms to make the classification whether the malware was using the Tor network for its operations or not. To assess the performance of the algorithms, multiple evaluation procedures and metrics were used. The exact figures are presented in section four. The programs created to make these experiments were collectively named "RETOMOS"-an acronym for "Recogniser of Tor Malware and Onion Services". The source code available on: https://github.com/jesperatstockholmuniversity/RETOMOS

3.1 Experiment setup and its limitations

The experimental setup for this research study consisted of two physical machines and three modules, as depicted in Fig. 2. There was one physical machine for running the Cuckoo sandbox, and one for running the scripts of the RETOMOS toolset. Each machine contained one or two modules, namely the following.

1. Cuckoo Sandbox Machine
   - The malware behaviour analysis system that logged the behaviour of the executed malware samples and wrote structured malware report files from them.
2. RETOMOS Machine
   - The malware behaviour report database that contains "washed down" malware reports that were fetched from the Cuckoo sandbox machine via rsync.[2] A script of the RETOMOS toolset was used to extract the features from the Cuckoo report files into the database. The database itself was used as training data for the classifiers.
   - The machine learning based classifier - a program that used the malware behaviour reports as training data to make classifications and identify.onion URLs from them. Writes the result in an output file.

The hardware and software setup was similar to the one presented by Pirscoveanu et al. [39], that also used Cuckoo Sandbox to extract behavioural log files and then analyse them using machine learning algorithms. The RETOMOS scripts used to extract data from log files, wash the data, inserting it to a database, and then finally analysing the data to train classifiers on them are available on the researchers' Github.com page.[3] The RETOMOS repository contained the following scripts:

- `retomos.py` - This script is the main program that receives the user input and runs the below scripts with the user specified options.
- `retomos_feature_extractor.py` - This script extracts.json data from malware reports and inserts it into an SQLite3 database to use it as training data later on.
- `retomos_malware_classifier.py` - This script trains the classifiers on the SQLite3 database input and tries to extract.onion URLs from malware samples classified as Tor dependant.

3.1.1 Malware samples

The malware samples used in the experiment were downloaded from two repositories on [59]. All files that were not .exe (header 4A 5A) files were excluded from the analysis. The full list of the 4794.exe only malware samples can be found on: https://github.com/jesperatstockholmuniversity/RETOMOS/blob/main/All_EXE_Malware_Samples.sha256

```
File: VirusShare_00357
MD5: 88228ea666e518190a935e92b03481e1
SHA1: 304bfcec1162073aea4ecaeaea2a4813
80405283

File: VirusShare_CryptoRansom_20160715.zip
MD5: 451f9041eb6c0c585bdc61ed9f8aefbf
SHA1: cecc29daa705207ac407921b09a97786
21fc3733
```

Virusshare.com is a website run by Corvus Forensics[4] which has been used in other machine learning focused malware research studies: [1, 3, 25, 39]. The files in the repositories provided by Virusshare.com are of mixed formats, and for this research study, all files that were not .exe files were excluded in order to have as homogeneous input data as possible, in order to make guarantee that all files were analysed in the same way in Cuckoo sandbox.

3.1.2 Cuckoo Sandbox setup

A Dell XPS with an Intel I7 CPU, 12 GB of RAM, running Ubuntu 20.04 and the malware analysis environment Cuckoo Sandbox 7.2 [5] under Python 2.7, KVM and QEMU was used as the choice of virtualisation software for the Cuckoo analysis machine. The "guest" or the "victim" machine was a virtual Windows 7 32 bit machine with 4 GB of RAM, and the software installed to simulate an authentic Windows machine such as Tor Browser, Skype, Chrome, Firefox, Slack, Acrobat Reader, as well as .jpeg and .png pictures, .pdf files, and .exe files in the user directory of the default user.

Cuckoo sandbox by default runs certain modules for analysing behaviour and registering meta data, but also offers multiple modules to be additionally enabled in the victim machine to assess how the malware behaves when run in a live system. The available modules were the following for Cuckoo version 2.0.7 [10].

- **applet**: used to analyze Java applets.

[2] https://rsync.samba.org.

[3] https://github.com/jesperatstockholmuniversity/RETOMOS.

[4] https://corvusforensics.com/.

[5] https://cuckoosandbox.org/.

- **bin**: used to analyze generic binary data, such as shellcodes.
- **cpl**: used to analyze Control Panel Applets.
- **dll**: used to run and analyze Dynamically Linked Libraries.
- **doc**: used to run and analyze Microsoft Word documents.
- **exe**: default analysis package used to analyze generic Windows executables.
- **generic**: used to run and analyze generic samples via cmd.exe.
- **ie**: used to analyze Internet Explorer's behavior when opening the given URL or HTML file.
- **jar**: used to analyze Java JAR containers.
- **js**: used to run and analyze Javascript files (e.g., those found in attachments of emails).
- **hta**: used to run and analyze HTML Application files.
- **msi**: used to run and analyze MSI windows installer.
- **pdf**: used to run and analyze PDF documents.
- **ppt**: used to run and analyze Microsoft PowerPoint documents.
- **ps1**: used to run and analyze PowerShell scripts.
- **python**: used to run and analyze Python scripts.
- **vbs**: used to run and analyze VBScript files.
- **wsf**: used to run and analyze Windows Script Host files.
- **xls**: used to run and analyze Microsoft Excel documents.
- **zip**: used to run and analyze Zip archives.

In the experiment setup used for this research study, the following process modules were used: "generic", "bin", "exe", "dll". These modules were considered the most suitable ones for analysing malicious.exe files, since the computational cost to include other, above mentioned, modules as well would be too high, and only .exe files (with header 4D 5A) were submitted sequentially for analysis via the Cuckoo API.

The routing option was set to "none", which means that Cuckoo does not route the traffic in any way; the routing is done as it is configured to in the operating systems. The platform to "windows", since the victim analysis virtual machine was a Windows 7 system. The time limitation for running each malware sample was set to 320 s (circa 5 min and 19 s). The default timeout for Cuckoo is 120 s [9].

Cuckoo supports connecting the analysis "victim" virtual machine to the Tor network, though this option is highly discouraged in order not to put too much unnecessary pressure on the Tor network [11]. Routing the malware analysis traffic via Tor would also delay the analysis process.

These process module behaviours were registered by Cuckoo and written to report logs files that were later extracted and used as features in the training data sets to train the different classifiers. Listing 1 presents an excerpt from a Cuckoo report file. Cuckoo logs all everything a malware sample does inside the sandbox/victim machine: network activity, Windows registry activity, used DLLs, used API calls, and more [10].

Listing 1 Excerpt from Cuckoo behaviour analysis report


```
"target": {
    "category": "file",
    "file": {
        "yara": [],
        "sha1": "25f28f48599759dfb8b6df88
            b13d93ba0610fa9a",
        "name": "VirusShare_c1242905e1c0
            af38d0d8f4c5a3dbbdc6",
        "type": "HTML document, Non-ISO extended-
            ASCII text, with very long lines, with
             CRLF line terminators",
        "sha256": "e5484a01adf42d668fc51b02b16
            c0b86cddbe3f91fd0178011271 c73bf51d85f
            ",
    "urls": [
        "http://my.jiehun.cn",
        "http://img1.jiehun.cn/image/sd_nav.gif",
        "http://img1.jiehun.cn/image/sousuoanniu.
            gif",
        "http://shop.jiehun.cn",
[...]

{
    "markcount": 34,
    "families": [],
    "description": "File has been identified by 34
        AntiVirus engines on VirusTotal as
        malicious",
    "severity": 5,
    "marks": [
    {
        "category": "Bkav",
        "ioc": "JS.eIframeRedirectNMe.",
        "type": "ioc",
        "description": null
    },
    {
        "category": "MicroWorld-eScan",
        "ioc": "Trojan.Script.503239",
        "type": "ioc",
        "description": null
    },
    {
        "category": "CAT-QuickHeal",
        "ioc": "HTML.Redirector.J",
        "type": "ioc",
        "description": null
    },
```

3.1.3 RETOMOS setup

The RETOMOS environment in which the data analysis and the classification tasks were programmed and run was a 64-bit 8 core AMD FX-8350, 16 GB DDR3 (non-ECC) RAM, running Arch Linux 5.13.12 and Python 3.9.6 with Scikit-learn version 0.24.2. The behavioural analysis reports generated by the Cuckoo server were downloaded to the analysis computer via rsync. Then they were "washed" and imported into an SQLite3 database using the RETOMOS script `retomos_feature_extractor.py` to become the input data set for the classifiers. By using Scikit-Learn's libraries, the classification models were then built using the database input in the script `retomos_malware_classifier.py`. The classification results were evaluated for recall, precision, f1, and accuracy scores using a 5-fold cross-validation for each classification algorithm. The next sections describe in detail how this was done.

3.1.4 Tor malware data set

Since there was no data set off-the-shelf, the alternative was to create a set for training the machine learning classification models. According to [22], 55.6% of the data sets used in digital forensic research were created by the researchers themselves. Out of a total of 4794 malware samples analysed, 72 ransomware samples using onion sites for their intended operational behaviour were identified and assembled along with 78 pseudo-randomly selected samples that did not rely on Tor using SQLite3 random()[6] function and verifying that the randomly selected samples were not part of the 72 confirmed Tor related malware samples.

The reason for using only 78 of the non-Tor related malware samples was the need for a balanced data set. Practically the data set used could have consisted of 72 Tor related and 4722 non-Tor related samples respectively, though that would have resulted in an imbalanced data set for training classification models. The 72 samples with the Tor flavour included Tor proxy connection attempts such as `sdfnsdkfndfgkjgnf.onion.to.com`. These class labels were identified by manual inspection of wide-scope keyword searches on the behavioural reports generated by Cuckoo. In practice, this was done running using the GNU grep command in Linux:

```
$ grep -i ".onion\|onion\|
tor browser\|tor_browser" *.json
```

Since Cuckoo behaviour reports contain a vast amount of information about network connections, registry entries, DLLs, API calls and other system and program execution details, it was simple, yet effective to make clear text searches on the 4794 report files to find certain activities based on keywords. The chosen keywords ".onion", onion", "tor", "tor browser", and "tor_browser" were generic but effective and resulted in 340 number of hits in the collection of 4794 malware analysis reports. 268 of these were false positives and were hence excluded from the Tor malware data set; 72 were kept.

It was deemed better to manually exclude the false positives than risk missing out the true positives. In addition, the malware's usage of registries entries was manually inspected to identify malware that uses the Tor browser for its operations. The malware names given by the AV vendors were also inspected to determine whether the malware sample used to the Tor network for its operations. The grep key word search is a very generic and primitive Tor related malware identification, hence manual inspection by the researchers was needed to exclude the behavioural log files that contained the keywords but were not in any way relating to the Tor network. Each report with a connection to the Tor network was then cross-referenced with their labels (i.e. the names of the malware, given by the antivirus organisations) retrieved from VirusTotal.com, as a way to check whether or not the labels gave any indication of Tor dependent malware. The malware samples used in this research study were active around 2016. At that time the anti-malware company Kaspersky identified a certain Tor using malware family as "Trojan.Onion" [28]. Hence, Kaspersky's anti-malware labels were used as the AV labels in our data set as they were deemed to most accurately describe the possible Tor related samples in the malware repository used.

The class labels in the data set were binary with the Tor related malware having the value "yes" (1), while in the opposite case the value was set to "no" (0). After creating the data set, the class labels were cross-validated by the two authors of this paper. The researchers manually went through all 150 malware reports and confirmed that they were correctly classified as either Tor related or non-Tor related based on the content of each behavioural analysis log file. The Tor and non-Tor malware data set is available on the researchers' Github.com page.[7] As mentioned, Cuckoo offers different analysis modules for analysing each malware sample utilised as input. All these modules log their analysis results in the behaviour report. These modules' report entries were extracted and inserted to the database (i.e. the training data set) and used as features (or attributes) for the Tor malware machine learning classification models, see example in Table 1. In order to not include any irrelevant features, the same ones as used in similar research articles [24, 39, 55], were inserted to the database, namely: DLLs, Mutexes,

[6] https://sqlite.org/lang_corefunc.html#random.

[7] https://github.com/jesperatstockholmuniversity/RETOMOS

Table 1 An example of the features and the Tor class label from the dataset created to train the Tor malware classifiers

ID. No.	AV Label	ASCII Strings	DLLs	Tor
integer	*string*	*string*	*string*	*boolean*
1	Win32.Trojan	MZ..	NTUserDat	0
..	..	..	..	..
150	Win32.Onion-Ransom	MZ..	NTGetProcess	1

DNSs, IPs, strings, and API calls. These mentioned features are generally the most important ones for malware analysis and are further described in the list below.

- **AV Label** - This was the malware sample name given to it by AV vendors. In this study we used the labels by Kaspersky.
- **ASCII Strings** - Strings found when the malware sample was run in the sandbox.
- **DLLs** - Dynamic link library used by the malware when run in the sandbox.
- **Mutexes** - Mutex objects used by the malware when run in sandbox.
- **API Calls** - API calls made by the malware when run in sandbox.
- **DNS** - Domain name system related artefacts registered from the malware when run in sandbox.
- **IP** - Internet protocol related artefacts registered from then malware when run in sandbox.

Together, these recorded malware characteristics create a wealth of data for the classifiers to learn and make predictions based on; they will not only look at DLLs to classify an executable, rather they will look at all the malware artefacts. Although the created training dataset appears to be a somewhat a small set, it was deemed sufficient for our experimental study considering the research deficit in the Tor flavoured malware. Furthermore, the data set will fit the purpose as an example and an indication of usefulness which amounts to a proof of concept.

3.1.5 Classification models

To build the classifiers, features from the SQLite database containing the behavioural details: Domain Name System (DNS) requests, API calls, and registry keys of each malware sample were fetched and inserted in a Pandas data frame that was then converted to a count vector.[8] The count vector held a matrix of counts of each malware sample's DNS requests, API calls, and registry key actions, i.e. the features of each sample. This count vector was then combined with the target (or class) labels of each sample, i.e. Tor (1) or non-tor (0) related.

To pre-process the data for the classification models to reduce the number of redundant features, Scikit Learn's feature selection algorithm VarianceThreshold was used.[9] All features with variance lower than 0.1 were set to be removed. Once, the feature selection was done the data set was split into a training set and a testing set. Scikit Learn's [37] built in training/test division algorithm was used to create the training and test data sets splitting them up in a division of 70% and 30% respectively.

Once the training and test data sets were divided, the classification models Logistic Regression (LR), Support Vector Machine (SVM), Multinominal Naive Bayes (MNB), Random Forest (RF), and Decision Tree (DT) were trained on them. All of these algorithms were part of the Scikit-Learn library [37].

The multinomial naïve Bayes (MNB) algorithm is an implementation of Bayes' theorem tailored for multinomially distributed data. It calculates the probabilities of features that belong to a specific class based on the characteristics of each class, combined with prior probabilities. MNB is an algorithm commonly used in text classification [33, 47].

Support vector machine (SVM), is an algorithm that linearly separates classes in a graph, a *linear discriminant*, and calculates the margin between the separated classes and draws the line in the middle of them [42, p. 92].

Logistic regression (LR) is a linear prediction classifier that is not by definition a regression, but a probabilistic classifier that calculates the membership probability of a certain class based on log-odds calculation [14, 48].

Random forest (RF) is an ensemble learning algorithm that constructs a collection of randomised tree-structured classifiers [27]. The randomness in combination with multiple models, makes RF less sensitive to overfitting.

A decision tree (DT) classifier is constructed as a tree that splits into leaves with features and classes with a calculated probability for the class affiliation with a specific leaf feature [45].

[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[9] https://scikit-learn.org/stable/modules/feature_selection.html#variance-threshold.

Table 2 Precision and recall scores of the multinomial naïve Bayes Tor malware classifier

	Precision	Recall	F1 score	Support
Tor	0.56	1.00	0.71	15
Non-Tor	1.00	0.29	0.45	17

Table 3 Precision and recall scores of the support vector machine Tor malware classifier

	Precision	Recall	F1 score	Support
Tor	0.80	0.80	0.80	15
Non-Tor	0.82	0.82	0.86	17

Table 4 Precision and recall scores of the logistic regression Tor malware classifier

	Precision	Recall	F1 score	Support
Tor	0.94	1.00	0.97	15
Non-Tor	1.00	0.94	0.97	17

3.1.6 Evaluation of the classifiers

Each classification algorithm's performance rate was evaluated using a 5-fold cross-validation score function in Scikit.[10] Accuracy, precision, recall, and F1 scores of the classifiers were calculated based on the metrics library from SciKit.[11] The same library was at our the disposal to calculate the receiver operating characteristic (ROC) and the area under the ROC curve.

The features were comprised of strings such as DNS queries, registry keys, and API calls. The class labels were binary (yes/no) and the samples were balanced that is 48% and 52% of each class respectively. Hence, accuracy could be considered as an adequate measure for the performance of the classifiers, unlike the case for imbalanced data sets where accuracy could be a misleading measure [27].

To assess the performance of the developed toolset, hyperfine[12] was used to calculate the execution times for the entire program over multiple runs. Hyperfine was configured to execute the program 10 times and calculate the time differences between the different executions to produce statistics from it.

[10] https://scikit-learn.org/stable/modules/cross_validation.html.

[11] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.

[12] https://github.com/sharkdp/hyperfine.

Table 5 Precision and recall scores of the random forest Tor malware classifier

	Precision	Recall	F1 score	Support
Tor	0.80	0.80	0.80	15
Non-Tor	0.82	0.82	0.82	17

Table 6 Precision and recall scores of the decision tree Tor malware classifier

	Precision	Recall	F1 score	Support
Tor	0.94	1.00	0.97	15
Non-Tor	1.00	0.94	0.97	17

3.1.7 Experiment setup limitations

On one hand, Cuckoo sandbox uses virtual machines to execute the software for the analysis. On the other hand, it is worth noting that some malware uses virtual environment recognition techniques to avoid analysis. Therefore, there were samples which yielded false negatives when run in the Cuckoo environment. In our data set, only true positives, i.e. confirmed malware samples, were included. The same goes for the malware that used packers and other evasion techniques to avoid detection by malware analysis systems. Some malicious software is configured not to run until a certain point in time, therefore such malware samples could not effectively infect the victim machine and execute properly since there was a time limitation for each sample of 320 s (circa 5 min 19 s). Two minutes is the default timeout in Cuckoo sandbox and was considered too little, thus a longer, yet reasonable timeout was configured for running each of the 4794 executables.

4 Results

The classification models were evaluated using the following metrics: accuracy, precision, recall, and F1 scores; receiver operating characteristic (ROC) curve and the area under the ROC curve. The classification performance of the logistic regression (LR) (see Table 4) and the decision tree (DT) (see Table 6) algorithms proved to be the most accurate compared the random forest (RF) (see Table 5), support vector machine classifier (SVM) (see Table 3) and the multinomial naive Bayes (MNB) (see Table 2). The 5-fold cross validation of the classifiers reached accuracy levels as follows.

- MNB: 56%
- SVM: 80%
- LR: 87.14%
- RF: 86.66%

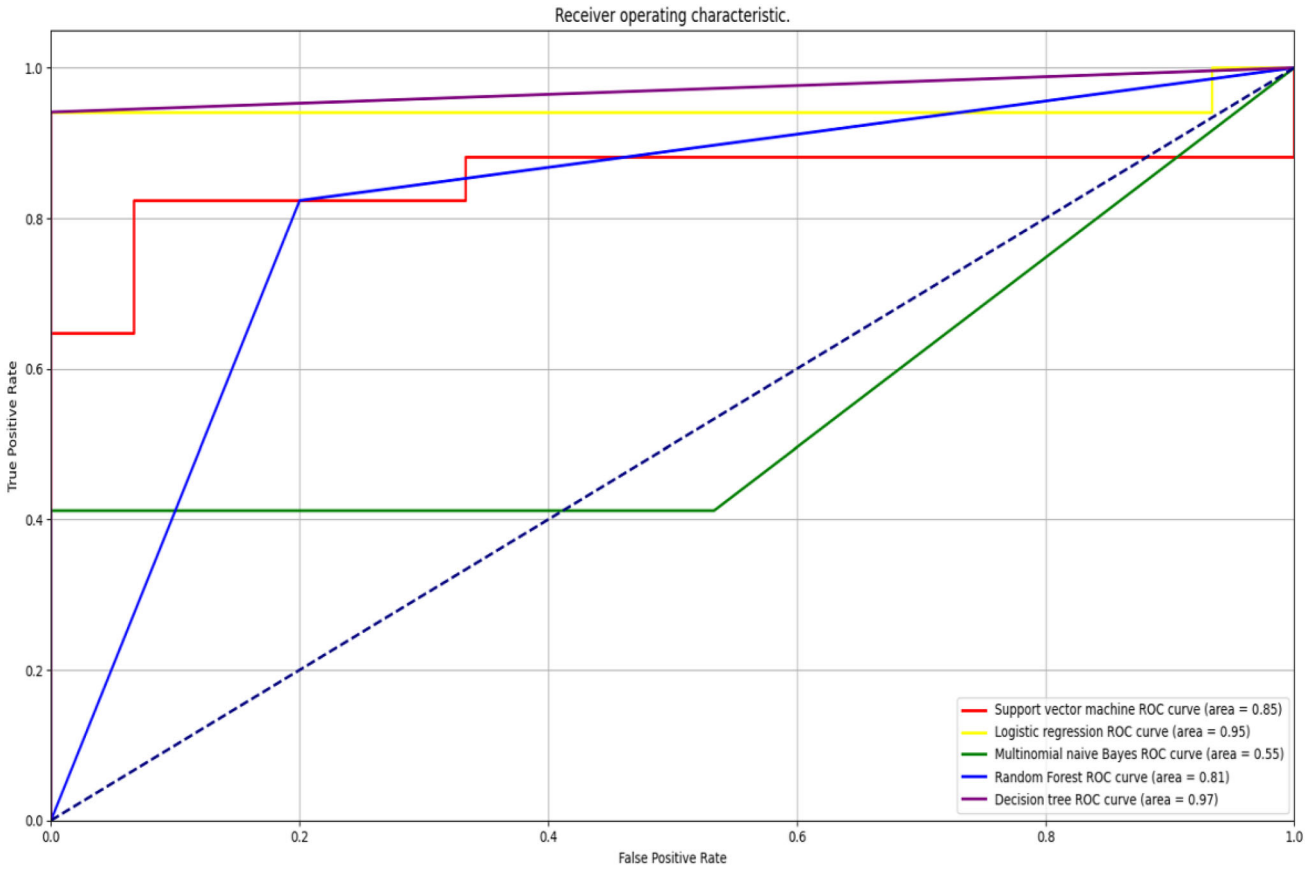

Fig. 3 Area under the receiver operator curve for the support vector machine classifier (red), logistic regression (yellow line), and multinomial naïve Bayes (green line), random forest (purple line), and decision tree (blue line)

- DT: 93.33%

The area under the receiver operating characteristic (ROC) curve reached a value of 0.85 for the support vector machine, 0.81 for the random forest, and 0.95 for the logistic regression, 0.97 for the decision tree, and 0.55 for the multinomial naïve Bayes classifier (see Fig. 3). In an ROC curve, the classifier's true positive predictions are depicted on the Y axis and the false positive predictions on the X axis; hence 1,0 (only true positives) are the ideal values for the classifier [42] [17]. The area under the ROC curve (AUC) is a calculated figure of the portion of the area of the unit square [17].

The computational performance evaluations of the classification tool, the script named `retomos_malware_classifier.py` in the developed toolset, took 560 s on average. Over 10 runs, the classification took 560 s ± 13.26 s with a minimum run time of 549 s and a maximum of 593 s.

```
$ hyperfine "python retomos.py -d
tor_and_non-tor_malware.db"

Time (mean sigma ): 560.071 s
 13.260 s
     [User: 451.033 s, System:
 110.172 s]
 593.375 s
10 runs
```

5 Discussion

The experiment results indicate that the classification models could, successfully, distinguish Tor dependent malware from non-Tor dependent malware based on their respective behaviour. The main contribution of the research is not only the sufficient evaluation scores; the clear indications of the effectiveness of automatically analysing malware to identify new Onion services subject to, for example, cyber threat intelligence research, is an equal, if not more evident, contribution.

The decision tree (DT) and the logistic regression (LR) classifiers were the winners amongst the five considered classifiers. The DT and the LR classifiers reached the same precision and recall for both the Tor related and the non-Tor related samples: 0.94 and recall 1.00 for the Tor related

malware, and vice versa 1.00 precision and 0.94 recall for the non-Tor dependant malware. The harmonised mean, the F1 score, for both LR and DT was 0.97, which should be considered rather high. However, the five-fold cross validation accuracy score for the DT classifier was higher than the LR score: 0.93 and 0.87 respectively. Neither the random forest, support vector machine, nor the multinomial naive Bayes were as precise in its predictions. The MNB achieved an inferior accuracy score of 56% on a five-fold cross validation, which is not very surprising, since it is one of the most rudimentary classification algorithms. However, the MNB serves the purpose of a reference point in comparison with the other, more sophisticated algorithms. The DT classifier reached an accuracy level of 93%, which is a sufficient accuracy level. The accuracy score is calculated by taking the number of correct classifications divided by the number of total classifications. The accuracy score is an adequate metric of the performance of a classifier given that the data set is balanced, i.e. having an even distribution of samples adhering to a specific class, as was the case in this research study [42, p. 190].

The five fold cross-validated accuracy scores were somewhat lower than the non-folded evaluations of the classifications. An indication that the data set consisted of some outliers that skewed the accuracy scores when not run with a dynamic training and test split. A larger training and testing data set rectifies the problem with the outliers.

Due to the fact that the class label of the used data set was binary, and that the balance between the classes was 72 respectively 78 samples of each class, accuracy is an applicable and acceptable measure for determining the success of the classifier. However, receiver operating characteristic (ROC) curve and area under the ROC curve (AUC) evaluation metrics are less sensitive to unbalanced data sets, and provide a more comprehensive measure of the performance of the classifier than scalar measures, such as accuracy or precision/recall [17, p. 873]. While the measures are not ideal, the accuracy and precision/recall values still indicate that the classifiers work reasonably well. The decision tree and the logistic regression classifiers proved to be the most successful ones, while the support vector machine, and the random forest classifiers performed at an acceptable level. The multinomial naive Bayes was just slightly better than random guessing.

Even though the data set is balanced and the number of features is low, the most adequate evaluation metric is the ROC curve and the area under the ditto. The AUC scores for both the DT and the LR classifiers are over 0.90, where 1.0 of course is the perfect [17]. What can be concluded from these figures is that the classifiers are likely to make a correct prediction on previously unseen, yet characteristically similar, data.

To identify the Tor dependant malware to include in the training data set, a keyword search using GNU grep was used. A simple keyword search on the set of 4794 Cuckoo behaviour reports resulted in 340 hits, 268 of these were not relevant (false positives) since they included the word "onion" or "tor" as part of their name such as "operator", "storage browser" which contains both "tor" and "browser", or a a foreign word like "chronione".

Although a keyword search was useful in identifying a wide scope of positives, both false and true ones, to create a small data set, it was not as effective in classifying Tor related and non-Tor related malware. The GNU grep keyword search yielded 72 true positives that were manually selected and included in the training data set of 150 samples in total. A total of 72 true positives in a collection of 340 positives equals a hit rate of 0.21, which is not to be considered effective, nor efficient.

To summarise: the presented classification models, unsurprisingly, vanquished the keyword search hit rate in our experiments. With respect of that, the benefit and motivation of using machine learning-based classifiers is clearly apparent.

The use case for this toolset is primary as an initial detection tool for finding new Tor malware and related onion services and .onion URLs that might need to be assessed by for example, law enforcement agencies. The malware and server assessment could be part of offensive and/or defensive operations, including intelligence gathering via server recognisance, monitoring, and scraping for cybercrime forensics and investigation.

However, the computational performance of the classifier in the toolset is low; in order for the toolset to be useful in a real-world scenario, it needs to be faster. This could be done by running the classification tasks in parallel on different CPU cores and fine-tuning the Python code.

Combined with a crawler, the toolset presented in this article is deemed as an effective technique for LEAs to identify new Tor related nefarious servers to monitor and collect information about. With a continuous supply of new malware samples that can be analysed in the sandbox environment, investigators can swiftly get hold of the latest Tor dependant malware and its possible related .onion service URLs.

5.1 Limitations

The data set was limited to 150 malware samples where 72 were Tor related malware samples and 78 were non-Tor related. While this might be classified as a small sample size, the results indicate that the classification model is more than sufficient, and could be subject to modifications and extensions in the future. On the contrary, each malware sample generates a lot of textual data in the form of analysis log file entries. Pirscoveanu et al. [39] reduced the number of

API calls by including only the 200 first API calls from the behavioural log in keeping the matrix to a "reasonable size".

6 Conclusion and future research

In this paper, the authors present a toolset called RETOMOS, that is capable of extracting features from malware behaviour log files and feeding them to five different machine learning classifiers out of which two of them succeeded in classifying Tor related malware at an accuracy level of over 90%.

By using a decision tree classifier, the authors have shown that it is possible to, with an accuracy of 93%, identify malicious software using the Tor network for its operations by analysing the behavioural data that is produced when running the malware in a sandbox environment without connecting to the Tor network.

The developed toolset proved useful and it might be used by law enforcement agencies to automatically analyse and identify new Tor-dependent malware and onion services, without having to utilise intense manual labour. By complementing the toolset with recognisance components, such as a server scanner or a scraper, it could also enable automatised monitoring and archiving of identified onion services and onion websites respectively.

The classification model and sandbox environment that comprise the toolset, should be combined and used to analyse and classify previously unseen malware. Accordingly, the Tor malware can be automatically identified at an early stage of the investigation so that the monitoring and the analysis could focus on the malware's related onion services. In a law enforcement agency setting, this could be a beneficial way of keeping up-to-date with the latest Tor facilitated malware and the correlating onion services, whose complex and unique URLs are difficult to find otherwise.

One of the future directions of the study might include an extension of the training data set to train the classifiers to look perpetually for new Tor malware to generate an even bigger data set to further improve the classifications and even more so the predictive potential of Tor malware. Another suggestion is to build in the classification models into the sandbox environment to create a quicker and more uniform Tor malware recognition system. Another direction worth exploring is to combine several algorithmic families such as random forests or bootstrap, less sensitive to bias and small data sets, to be used in particular cases for the initial phase of investigating Tor facilitated malware. Finally, cross-fertilisation with other paradigms that stem from the area of artificial intelligence, including neural networks and flavours of deep learning may offer not only alternative inference mechanisms, but also provide a basis for parallel reasoning as a way for mutually checking comparative, yet different, procedures in the investigation process.

Declarations

Some journals require declarations to be submitted in a standardised format. Please check the Instructions for Authors of the journal to which you are submitting to see if you need to complete this section. If yes, your manuscript must contain the following sections under the heading 'Declarations':

Funding Open access funding provided by Stockholm University.

Availability of data and materials Available via links provided in the article (removed for blind review)

Code Availability Available via links provided in the article (removed for blind review)

Declarations

Conflict of interest Check journal-specific guidelines for which heading to use: None

Ethics approval Not applicable

Consent to participate Not applicable

Consent for publication Not applicable

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Ahmed, Y.A., Kocer, B., Huda, S., et al.: A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection. J. Netw. Comput. Appl. **167**(102), 753 (2020). https://doi.org/10.1016/j.jnca.2020.102753
2. Alazab, M., Venkataraman, S., Watters, P.: Towards understanding malware behaviour by the extraction of api calls. In: Second Cybercrime and Trustworthy Computing Workshop. IEEE, Ballarat, Australia, pp. 52–59 (2010) https://doi.org/10.1109/CTC.2010.8
3. Alazab, M., Alazab, M., Shalaginov, A., et al.: Intelligent mobile malware detection using permission requests and api calls. Fut. Gener. Comput. Syst. **107**, 509–521 (2020). https://doi.org/10.1016/j.future.2020.02.002
4. Bernaschi, M., Celestini , A., Guarino, S., et al.: Spiders like onions: On the network of tor hidden services. In: The World Wide Web Conference. Association for Computing Machinery, New York, NY, USA, WWW '19, pp. 105–115, (2019) https://doi.org/10.1145/3308558.3313687

5. Casenove, M., Miraglia, A.: Botnet over tor: the illusion of hiding. 6th International Conference On Cyber Conflict (CyCon 2014) pp 273–282,(2014). https://doi.org/10.1109/CYCON.2014.6916408
6. Chen, H., Chung, W., Quin, J., et al.: Uncovering the dark web: a case study of jihad on the web. J. Am. Soc. Inf. Sci. Technol. **59**(8), 580 (2008)
7. CISA, Ransomware-what it is and what to do about it. (2019) https://www.us-cert.gov/sites/default/files/publications/Ransomware_Executive_One-Pager_and_Technical_Document-FINAL.pdf
8. Crowder, E., Lansiquot , J.: Darknet data mining–a canadian cybercrime perspective. (2021) arxiv:2105.13957
9. Cuckoo, cuckoo/test config.py at master cuckoosandbox/cuckoo. (2020) https://github.com/cuckoosandbox/cuckoo/blob/master/tests/test_config.py
10. Cuckoo.org Analysis package-cuckoo v2.7.0 book. (2019) https://cuckoo.sh/docs/usage/packages.html?highlight=module
11. Cuckoo.org , Pre-analysis network routing-cuckoo v2.7.0 book. (2020) https://cuckoo.sh/docs/installation/host/routing.html?highlight=tor#routing-tor
12. D'Agostino, V.D.: Complaint: United States of America v. blake benthall. (2014) https://www.justice.gov/usao/nys/pressreleases/November14/BlakeBenthallArrestPR/Benthall%2C%20Blake%20Complaint.pdf
13. Dingledine, R., Mathewson, N., Syverson , P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium (2004)
14. Dreiseitl, S., Ohno-Machado, L.: Logistic regression and artificial neural network classification models: A methodology review. J. Biomed. Inf. **35**, 352–359 (2002). https://doi.org/10.1016/S1532-0464(03)00034-0
15. Europol: Drugs and the darknet: Perspectives for enforcement, research and policy. (2017) https://www.europol.europa.eu/publications-documents/drugs-and-darknet-perspectives-for-enforcement-research-and-policy
16. F-Secure : Ransomware timeline 2010–2017. (2017) https://blog.f-secure.com/ransomware-timeline-2010-2017/
17. Fawcett, T.: An introduction to ROC analysis. Pattern Recogn. Lett. **27**(8), 861–874 (2006)
18. Frank, J.M.R.: Sentiment crawling: extremist content collection through a sentiment analysis guided web-crawler. In: IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE (2015)
19. Gandotra, E., Bansal, D., Sofat, S.: Malware analysis and classification: a survey. J. Inf. Secu. **5**(2), 458 (2014). https://doi.org/10.4236/jis.2014.52006
20. Ghafir, I., Svoboda, J., Prenosil, V.: Tor-based malware and tor connection detection. In: International Conference on Frontiers of Communications, Networks and Applications (ICFCNA), pp. 1–6. (2014) https://doi.org/10.1049/cp.2014.1411
21. Gheorghe, A.: New backdoor allows full access to mac systems, bitdefender warns. (2016) https://www.bitdefender.com/blog/labs/new-mac-backdoor-nukes-os-x-systems/
22. Grajeda, C., Breitinger, F., Baggili, I.: Availability of datasets for digital forensics and what is missing. Dig. Investig. **22**(5), 594–5105 (2017). https://doi.org/10.1016/j.diin.2017.06.004
23. Horejsi, J.: Retefe banking trojan targets UK banking customers. (2016) https://blog.avast.com/retefe-banking-trojan-targets-uk-banking-customers
24. Hwang, J., Kim, J., Lee, S., et al.: Two-stage ransomware detection using dynamic analysis and machine learning techniques. Wireless Pers. Commun. **112**(112), 2597–2609 (2020). https://doi.org/10.1007/s11277-020-07166-9
25. Irshad, A., Dutta, M.K.: Identification of windows-based malware by dynamic analysis using machine learning algorithm. In: Gao, X.Z., Tiwari, S., Trivedi, M.C., et al. (eds.) Advances in Computational Intelligence and Communication Technology, pp. 207–218. Springer, Singapore (2021)
26. Juarez, M., Afroz, S., Acar, G., et al.: A critical evaluation of website fingerprinting attacks. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, CCS '14, pp. 263–274 (2014) https://doi.org/10.1145/2660267.2660368
27. Karlsson, I.: Order in the random forest. PhD thesis, Stockholm University, Kista, Sweden, (2017) https://su.diva-portal.org/smash/get/diva2:1090364/FULLTEXT01.pdf
28. Kaspersky (2022) The onion ransomware (encryption trojan). https://usa.kaspersky.com/resource-center/threats/onion-ransomware-virus-threat
29. Kwon, KH., Priniski, JH., Sakar, S., et al.: Crisis and collective problem solving in dark web: An exploration of a black hat forum. In: Proceedings of the 8th International Conference on Social Media & Society Article No. 45. ACM, pp. 1–5 (2017)
30. Ligh, M.H., Adair, S., Hartstein, B., et al.: Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code, 1st edn. Wiley, London (2011)
31. Ligh, M.H., Case, A., Levy, J., et al.: The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory, 1st edn. Wiley, London (2014)
32. Ling, Z., Luo, J., Wu, K., et al.: Torward: discovery, blocking, and traceback of malicious traffic over tor. IEEE Trans. Inf. Forensics Secur. **10**(12), 2515–2530 (2015)
33. Manning, C., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008). https://doi.org/10.1017/CBO9780511809071.014
34. Meng, Y., Zhuang, H., Lin, Z., et al.: A survey on machine learning-based detection and classification technology of malware. In: 2021 International Conference on Computer Information Science and Artificial Intelligence (CISAI), pp. 783–792, (2021) https://doi.org/10.1109/CISAI54367.2021.00158
35. Nunes, E., Diab, A., Gunn, A., et al.: Darknet and deepnet mining for proactive cyber treat intelligence. In: Conference on Intelligence and Security Informatics (ISI). IEEE, pp. 7–12, (2016) https://doi.org/10.1109/ISI.2016.7745435
36. Page, C.: Revil ransomware group goes dark after its tor sites were hijacked. (2021) https://techcrunch.com/2021/10/18/revil-ransomware-group-goes-dark-after-its-tor-sites-were-hijacked/
37. Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
38. Pektas, A., Acarman, T.: Ensemble machine learning approach for android malware classification using hybrid features. In: Proceedings of the 10th International Conference on Computer Recognition Systems. Springer, pp. 191–200 (2017)
39. Pirscoveanu, RS., Hansen, SS., Larsen, TMT., et al.: Analysis of malware behavior: type classification using machine learning. In: 2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA). IEEE, pp. 113–118, (2015) https://doi.org/10.1109/CyberSA.2015.7166115
40. Popov, O., Bergman, J., Valassi, C.: A framework for a forensically sound harvesting the dark web. In: CECC 2018: Proceedings of the Central European Cybersecurity Conference 2018. ACM, pp. 1–7, (2018) https://doi.org/10.1145/3277570.3277584
41. Portnoff, RS., Afroz, S., Durrett, G., et al.: Tools for automated analysis of cybercriminal markets. In: International World Wide Web Conference Committee (IW3C2). ACM, pp. 1–5, (2017) https://doi.org/10.1145/3038912.3052600
42. Provost, F., Fawcett, T.: Data Science for Business-What You Need to Know about Data Mining and Data-Analytic Thinking, 1st edn. O'Reilly, USA (2013)

43. Rabaut, JT.: Complaint: United states of america v. alexandre cazes. (2017). https://www.justice.gov/opa/press-release/file/982821/download
44. Reed, T.: New mac backdoor malware: Eleanor. (2016) https://blog.malwarebytes.com/cybercrime/2016/07/new-mac-backdoor-malware-eleanor/
45. Rokach, L., Maimon, O.: Decision Trees. World Scientific Publishing, Singapore (2010)
46. Saleem, J., Islam, R., Kabir, MA.: The anonymity of the dark web: a survey. IEEE Access 10:33,628–33,660. (2022). https://doi.org/10.1109/ACCESS.2022.3161547
47. Scikit-Learn-Developers: 1.9.2. multinomial naive bayes. (2020) https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes
48. Scikit-Learn-Developers: 1.1.11. logistic regression. (2022) https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic%20regression#sklearn.linear_model.LogisticRegression
49. Shalaginov, A., Banin, S., Dehghantanha, A., et al.: Machine learning aided static malware analysis: a survey and tutorial. Adv. Inf. Sec. **70**, 559 (2018). https://doi.org/10.1007/978-3-319-73951-9_2
50. Sophos: The current state of ransomware: Cryptowall. (2015) https://news.sophos.com/en-us/2015/12/17/the-current-state-of-ransomware-cryptowall/
51. Spitters, M., Klaver, F., Koot, G., et al.: Authorship analysis on dark marketplace forums. In: European Intelligence and Security Informatics Conference. IEEE, pp. 631–641, (2015)
52. StatCounter: Desktop operating system market share worldwide aug 2021–aug 2022. (2022), https://gs.statcounter.com/os-market-share/desktop/worldwide
53. Tai, XH., Soska, K., Christin, N.: Adversarial matching of dark net market vendor accounts. In: KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. IEEE, pp. 1871–1880, (2019) https://doi.org/10.1145/3292500.3330763
54. Tarakanov, D.: The inevitable move-64-bit zeus enhanced with tor. (2013) https://securelist.com/the-inevitable-move-64-bit-zeus-enhanced-with-tor/58184/
55. Tian, R., Batten, L., Islam, R., et al.: An automated classification system based on the strings of trojan and virus families. In: 2009 4th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, pp. 23–30, (2009)
56. Tor-Project: dir-spec.txt-torspec-tor's protocol specifications. (2022a) https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt
57. Tor-Project: glossary.txt - torspec - tor's protocol specifications. (2022b) https://gitweb.torproject.org/torspec.git/tree/glossary.txt
58. Tor-Project "tor rendezvous specification - version 3". (2022c) urlhttps://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt
59. Virusshare.com Virusshare.com-because sharing is caring. (2020) https://virusshare.com/research
60. Westlake, B., Bouchard, M., Frank, R.: Assessing the validity of automated webcrawlers as data collection tools to investigate online child sexual exploitation. Sexual Abuse **29**(7), 685–708 (2015). https://doi.org/10.1177/1079063215616818
61. Zainor, F.: Trojan-spy:js/retefe description. (2019) https://www.f-secure.com/v-descs/trojan-spy_js_retefe.shtml

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.