SN Computer Science (2022) 3:53
https://doi.org/10.1007/s42979-021-00926-9

ORIGINAL RESEARCH

Investigating Malware Propagation and Behaviour Using System and Network Pixel-Based Visualisation

Jacob Williams[1] · Phil Legg[1]

Received: 29 July 2021 / Accepted: 4 October 2021

Abstract

Malicious software, known as malware, is a perpetual game of cat and mouse between malicious software developers and security professionals. Recent years have seen many high profile cyber attacks, including the WannaCry and NotPetya ransomware attacks that resulted in major financial damages to many businesses and institutions. Understanding the characteristics of such malware, including how malware can propagate and interact between systems and networks is key for mitigating these threats and containing the infection to avoid further damage. In this study, we present visualisation techniques for understanding the propagation characteristics in dynamic malware analysis. We propose the use of pixel-based visualisations to convey large-scale complex information about network hosts in a scalable and informative manner. We demonstrate our approach using a virtualised network environment, whereby we can deploy malware variants and observe their propagation behaviours. As a novel form of visualising system and network activity data across a complex environment, we can begin to understand visual signatures that can help analysts identify key characteristics of the malicious behaviours, and, therefore, provoke response and mitigation against such attacks.

Keywords Malware analysis · Data visualisation

Introduction

Malware (malicious software) is recognised to be one of the greatest threats against modern computer systems that now underpin much of society and our everyday lives [1]. Recent years have seen a number of high profile cyber attacks on global corporations, resulting in significant financial, operational, and reputational damage [7]. In particular, ransomware has become a dominant malware variant due to advances in anonymous payment methods, such as Bitcoin, meaning that a victim's files can be encrypted and held to ransom until a user makes payment to acquire the decryption key. In addition, malware distribution mechanisms have evolved as part of our hyper-connected society. Vulnerabilities such as the MS17-010 EternalBlue exploit reveal how malware can propagate through corporate and home networks at speed and without user intervention which was a key software component to the propagation of the global WannaCry attack [24]. The arms race of security continues between security analysts and malicious software developers. As malware developers seek to identify new methods of attack, those tasked with defending systems need improved tooling to be able to examine patterns of behaviour effectively and efficiently to assist well-informed incident response.

A major complexity with any systems analysis is being able to examine large volumes of multi-variate data in conjunction. Reverse engineering techniques and static code analysis have often been used to examine malware characteristics. It is also increasingly common to perform dynamic analysis using a sandbox environment, which is the approach we adopt, however in a large sandbox network environment, a vast amount of machine data can easily be generated in a relatively short time, and there is a need for improved methods of analysis to understand this data at scale.

In this work, we investigate the nature of malware propagation using a virtualised multi-machine network

This article is part of the topical collection "Cyber Security and Privacy in Communication Networks" guest edited by Rajiv Misra, R K Shyamsunder, Alexiei Dingli, Natalie Denk, Omer Rana, Alexander Pfeiffer, Ashok Patel and Nishtha Kesswani.

✉ Phil Legg
Phil.Legg@uwe.ac.uk

[1] University of the West of England, Coldharbour Lane, Bristol BS16 1QY, UK

Published online: 09 November 2021

environment. In particular, we propose novel pixel-based visualisation techniques to help address the challenges of data scalability to improve the diagnostic of system and network behaviours to examine the characteristics of malware propagation across computer networks. In this manner, we are not only able to understand the traits that impact on an individual machine, but we can examine the traits of the overall network, including the propagation characteristics of how malware traverses from one machine to the next. A key challenge for any large scale analytics such as this is the issue of scalability, and therefore, we seek to identify techniques that can help gather and analyse large volumes of multi-variate data sources that contextualise the malicious activity across a network, in a manner that is scalable for an analyst to work with. Our focus is on the visualisation of malware propagation to enable security analysts a clear overview of how machines interact and respond to a malicious software threat. Using two well-recognised malware variants, and a sandbox network environment, we illustrate how these variants are visualised and how a security analyst can clearly identify key activities that result in malware propagation.

The contributions of this work are as follows:

- We propose pixel-based visualisation techniques for system and network-wide analysis, offering scalable representation of multi-variate system parameters.
- We deploy a testing environment and illustrate the use of these techniques to better convey the characteristics and behaviours of system and network activities, in a manner that allows visibility of all devices at scale.
- We illustrate how this can better examine propagation techniques in malware due to the comparative analysis of system properties across multiple networked devices.

Related Works

To understand the problem further, we focus our related works on the broad challenge of malware analysis, including traditional methods of analysis, machine-learning techniques, and visualisation methods. Previous methods largely focus on individual machine analysis. One of the key aspects of our work is to address multi-system examination and analysis simultaneously, such as a large corporate network or a home environment, such that we can better understand the visual cues and signatures of compromised devices through malware propagation methods that involve no human intervention. By focusing on the contributing elements of each paper, we are able to divide our research into four key areas of study, each of these were used to make decisions on how our development proceeded or changed as it progressed.

First and foremost was the research conducted into visualisation methods and how other malware analysts have visualised their findings in the past. Wagner et al. [26] surveyed various methods of visualising malware analysis, exploring the multitude of characteristics that apply to techniques for individual analysis, comparison, and summarization. Gregio et al. [11] explored the visualisation of malware behavioural analysis by capturing interactions with target operating systems using system service descriptor table hooking, using this data the authors were able to create visualisations that draw distinctions and similarities between samples. Han et al. [13] performed analysis of samples using visualised image matrices created by extracting opcode sequences from malware binaries through a combination of disassembly and execution of samples. In doing so, Han et al. [13] created visualisations that can be used to distinguish samples from each other and be used for training learning algorithms attempting to do the same. Similar to our initial goals, Gove and Deason [10] looked at identifying malware through network activity utilising an algorithm based on discrete Fourier transforms paired with aggregation summary tables which can be used to inform whether a detection is worth looking into. Similarly, Zhuo and Nadjin [29] conducted a study with the goal of visualising malware network traces by capturing heterogeneous attributes, such as protocols and IP addresses, an effective method due to the analysis of network traces being key to categorising malware. A study into visualising enterprise network attacks by Creese et al. [8] is important to our work as the result of their study was the creation of a visualisation tool for modelling attacks using traditional network notation and a disk propagation logic that connects the network and business process layers. Another key aspect of their work was to study the principles and characteristics they they identified during the development of their tool. Finally, a study into the design of pixel-oriented techniques by Keim [16] that shows how to create clear and effective pixel-based visualisations for communicating various forms of data, this was used to inform our design choices.

The field of malware detection is ultimately the product of effective analysis, and therefore, it is useful to study detection techniques to explore possibilities for visualisation. Using the method of mining format information from malware executables, Bai et al. [2] were able to identify characteristics that could be deemed malicious, identifying a total of 197 different features which can be used to train effective detection algorithms. NODENS is a random forest-based, lightweight malware detection platform created by Mills et al. [20], who explore the creation of classification models which use 22 different features curated after removing duplicate or unnecessary ones. Patel and Tailor [22] created a method for monitoring and counteracting ransomware by observing key folders and dynamically creating large dummy files when rapid encryption was detected within the observed

folders. In a similar vein to Grégio et al. [11], Xiao et al. [27] studied detecting malware through behaviour graphs which were created by monitoring the order in which malware makes system calls and in doing so created a method of distinguishing benign system call orders from malicious ones. Xiao et al. [27] also documented a broad scope of system calls that malware frequently make, which is useful for reproducing such work on classification. Donehue et al. [9] presented a method of detecting malware samples through the obfuscation method of packing using a tree based navigation method with applied offsets to distinguish what might be legitimate and what might be packing. In doing so, they created a technique that can also be visualised, Markov Byte Plots that show the difference between packed and unpacked portable executable files. Finally, Rhode et al. [23] studied the use of recurrent neural networks for predicting if an executable was malicious or not based on a short snapshot of behavioural data using values, such as total processes, max process ID, and CPU usage.

To create hypotheses and temper an expectation for what our system and visualisations were going to look like, it was sensible to explore research that focused solely on the propagation aspect of malware. Yu et al. [28] explored the concept of modelling malware propagation in large scale networks using a two-layer epidemic model. The authors discern that malware propagation in large networks can follow one of three different distributions depending on how much of the network is already infected. To better optimise intrusion detection systems, Sharafaldin et al. [25] created a data set containing more than eighty network traffic features gathered from live samples using CICFlowMeter, a publicly available ethernet traffic bi-flow generator and analyser. Using these features, the authors were able to create data sets and designate signatures to varying attack profiles. In a similar manner to Yu et al. [28], Hosseini and Azgomi [14] represented malware propagation using a rumour spreading model, identifying five different types of machines that perform different roles in the propagation of malware. Finally, Guillen and Rey [12] researched the role of compartment devices in malware propagation and found that while such devices do not contribute to the initial infection rate, they instead increase the general overall rate of infection.

The final area of study was researched to assist in the development of the sandbox environment that would be used to run and extract data from to fuel the visualisation process. Studies such as the one performed by Chakkaravarthy et al. [6] provided us with knowledge of common methods used to avoid detection in sandboxes, concepts, such as payload fragmentation, session splicing, and contextual awareness. The authors also provide a clear and detailed description of the setup that they used to conduct this study, making a useful baseline for replication. Afianian et al. [1] present a similar study, a comprehensive summary of knowledge regarding evasion techniques, separating them into the categories of detection dependent and detection independent, and detailing how to prevent both kinds. Finally, a study conducted by Miramirkhani et al. [21] looked deep into what small features could be used by malware writers to detect a virtual environment. They create distinctions between types of artefacts that one could look for, and subcategories within those artefacts. While the study we explore here does not seek to address sandbox evasion or detection, it is nevertheless important to recognise this important area of research.

Methodology

We first describe the experimental environment for our data collection, and then discuss the design aspects of conveying this data in a scalable manner for a security analyst.

Experimental Environment

Sandbox environments are commonly used for malware analysis [18], often achieved using a Virtual Machine to examine the behaviour of malware when executed. Cuckoo sandbox [15] is an excellent tool for malware analysis, however a limiting factor is that Cuckoo is designed for a single virtual environment rather than monitoring propagation of malware across machines. We, therefore, develop a custom environment based on Cuckoo but that can support and gather data from multiple targets.

Figure 1 shows our virtualised network consisting of four Windows 7 (Win7) nodes with IP addresses in the range 10.10.5.11-14 and a Ubuntu node with address 10.10.5.10. All devices are connected via a virtualised internal only network. We use VirtualBox as our virtualisation system as it offers excellent configuration options for virtualised networking, as well as offering the VBoxManage command line tools. The internal network is isolated, such that the Win7 hosts can interact with each other and the Ubuntu monitoring node, however there is no access back to the host machine to escape from the sandbox. The Ubuntu node is used for real-time data collection, and is air-gapped with two network adaptors meaning that if the Win7 devices are shut down, and the Ubuntu network adaptor connected to the internal network is disabled, then we can connect our host to this machine for the retrieval of data logs. Since many malware samples are designed to target Windows systems, using a Ubuntu environment for the monitoring system means that it is not exposed to the same vulnerabilities as the target machines being examined, but yet could be connected to the same virtualised network for the purpose of gathering and retrieving data logs safely. Furthermore, macOS was used as the host operating system for VirtualBox to limit any potential of the Win32 malicious samples escaping the

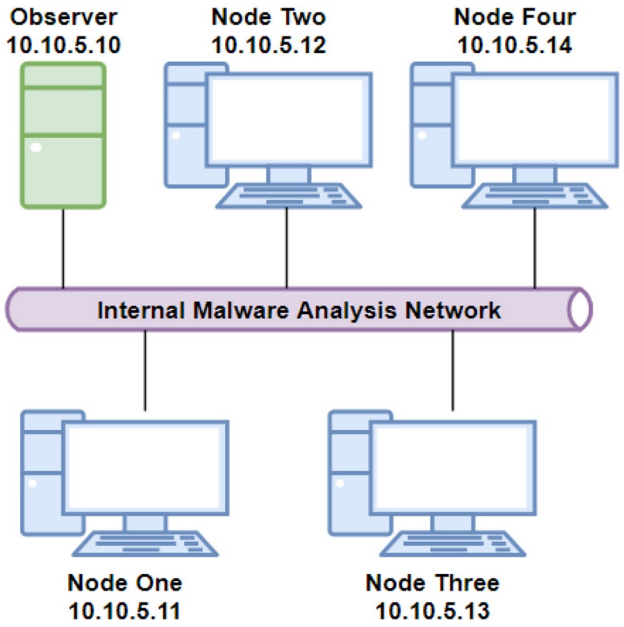

Fig. 1 Network Configuration. We host four virtualised Windows 7 machines for deployment of the attack. We also host a virtualised Ubuntu 20.04 machine for data collection. All machines are connected to each other on a internal virtualised network with no external access. External access can only be granted to the Ubuntu machine for the purpose of retrieving data collection

sandbox and causing damage to genuine production systems. The Win7 nodes were configured with various applications and utilities to represent real machines, including Skype, Office, Python, and VSCode. The machines were also deliberately weakened in terms of security, so that the true impact of propagation could be examined. Therefore, we disabled user access control and allowed all port connections to be accepted. We also allowed network discovery and file sharing across the networked devices.

For the data collection we utilise a variety of different sources across the network. First, using the Ubuntu node we gather system-specific data from each machine using a customised Python script, with the client script running on each of the four Win7 machines and a server script running on the Ubuntu 20.04 node. This script utilises the psutil Python library and queries each machine for general statistics related to CPU and RAM usage, network connections and system processes. Each client will report back to the server at given time intervals as specified by the user (e.g., every 15 s). This process is crucial as we can not store any logging data on the client machines themselves, since by the very nature of ransomware, these logs would become encrypted when the ransomware is deployed. The second mechanism in place for data collection is for network activity which we capture using Wireshark on the Ubuntu node. By enabling promiscuous mode on the Ubuntu network adaptor we can capture all traffic on the internal network, giving a complete view of all inbound and outbound traffic between the connected devices. Despite being isolated from the Internet, there is still significant information available here, as well as information related to the propagation of the malware over the network infrastructure. The third source of data collection is on the host OS running VirtualBox, where we can use the VBoxManage command line tools to obtain further detail on the virtual machines. This reports on CPU, RAM and HDD usage, as well as information on network connections (although due to being isolated this does not bring much information). In addition, we can use VBoxManage to capture screenshots of all virtual machines on a sequential basis. We configure this, so that we capture one screenshot of each machine every minute. The result of the data collection is that for every machine we have detailed time-stamped logs on CPU and RAM usage, screenshots, information on the network connections that are established and their protocols, and information on the system processes currently running.

For each test, we deploy the virtual machines and begin the logging process and allow the systems to run for a period of time with no further action, which is done to establish an initial baseline of activity. The Ubuntu node is then able to transfer the malicious test sample to the first Win7 node. We then execute the malicious sample under test on Node1 and we continue to observe and log the system and network activity. The duration of the test is dependent on the sample being tested, which could be from as little as 5 min to a couple of hours depending on what behaviours are performed. Crucially, the visual representation of the data needs to be scalable to account for variable length execution periods. Ransomware, by design, makes itself known to the user to inform them that their system is encrypted and that payment is required. Therefore, we can use the obtained screenshots to observe whether all nodes have been compromised. Since we are using virtualisation, we can then revert all of the Win7 virtual machines to their previous snapshot to reset the test environment. We can also then disconnect the Ubuntu node from the malware analysis virtual network to retrieve the data captures.

Visualisation Design

Security analysts are often tasked with understanding complex multi-variate data and related contextual data attributes using dashboard visualisations, such as charts and plots. A major challenge is being able to convey information in a concise yet informative manner, such that the context can be understood clearly while also being scalable for analysing multiple machines together, so that a holistic view can also be achieved. As described previously, the data collection results in a large multi-variate data set that includes CPU

and RAM usage, network connections, system processes, and screenshots, for each of the machines running on the network over an extended period of time. While traditional plots and charts may go some way to reveal temporal data variations, they require significant real estate on-screen and result in much white space. This does not lend itself to a scalable solution, where multiple machines and multiple data attributes can be easily examined together. As the amount of information to be represented is increased, it is important to consider the implications of visual aspects such as data-ink ratio and other cues that help make a visualisation engaging and memorable [5]. The smallest visual element available on a computer is a pixel. Pixel-based visualisations [16] are effective for conveying numerical data at scale, such as illustrating days of a year in a concise and compact format [4].

Figure 2 gives an overview of how we map data to the pixel-based visualisation. Each row represents a numerical data attribute (e.g., CPU usage, RAM usage, process usage, network protocol). Likewise, system screenshots are grouped by row to illustrate the system interface where required. The temporal data attributes are mapped against the corresponding screenshot for the same time period (e.g., 4 data observations captured every 15 s map against one screenshot captured per minute). Rows can then be stacked together as required, and colour-coded based on their respective system, with the data value represented by the size of the pixel within the square region. We can extend this further for data sources such as network activity to include source-destination pair as well as protocol and packet size. We consider each protocol as an independent data attribute, and we use colour to denote the destination machine, since known hosts are colour-coded and unknown hosts can be represented by grey scale. The value can be determined as either the number of occurrences for a given protocol, or an aggregated packet size for the given time period. In this manner, each row can be defined based on the machine, the data source, and any further attributes, such as protocol or process name (e.g., 'Node1-CPU-Usage', 'Node2-Network-SMB', 'Node3-Process-Skype'). Figure 3 illustrates in greater detail how each data point is mapped to the corresponding pixel region. A region of 9×9 means that we could in theory have 6 distinct square pixel sizes (9×9, 7×7, 5×5, 3×3, 1×1, and blank). Since we choose to map four data observations per screenshot, each screenshot is scaled to 36×36. We use a consistent thresholding approach across all data source values V based on the mean value M for the given data row. For each data row, if $V > 2 * M$ then the pixel is scaled as 9×9, else if $V > M$ then the pixel is scaled as 5×5, else if $V > 0$ then the pixel is scaled as 3×3, else the pixel region is left blank. We omit 7×7 and 1×1 to support a just-noticeable difference between the pixel sizes even at scale, however all numerical parameters can be easily modified by an analyst to support finer detail or to increase scalability, and summary statistics for each attribute row can also be examined in further detail. This is available for download to promote further research activity and collaboration with the wider community: https://github.com/phillegg/pixsys.

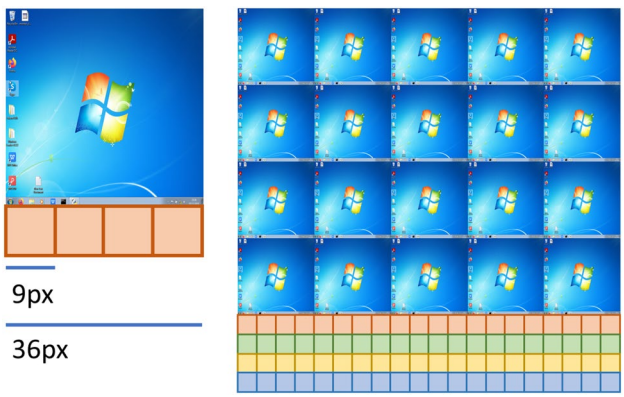

Fig. 2 Visual design overview. Here, one screenshot is mapped against 4 data observations of a single metric at 4 given time periods (e.g., every 15 s). This can be scaled up as shown, to represent 4 systems over a longer duration (e.g., 5 min)

Experimentation

We use two different malware variants to illustrate our analytical approach using the pixel-oriented visualisation technique: WannaCry and NotPetya. We specifically focus on these two samples as they both exhibit propagation using the EternalBlue exploit, yet they also have other differentiating characteristics, meaning we can examine these using our approach for where they may exhibit similarities and

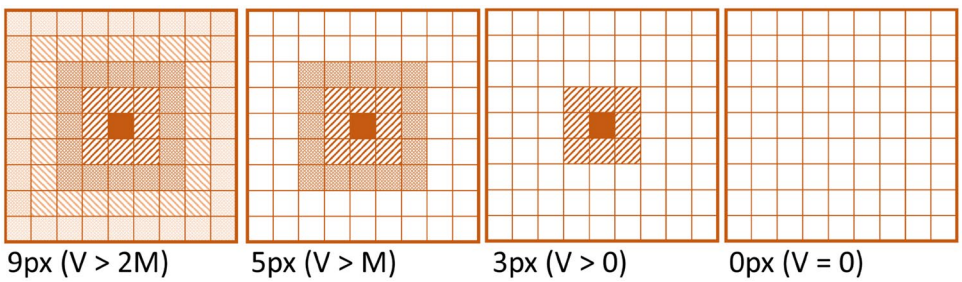

Fig. 3 Visual design of pixel representation using 4 sizes to support just-noticeable differences, based on whether the given value is greater than twice the mean, greater than the mean, greater than zero, or zero

differences. As described previously, we deploy a virtualised network consisting of four networked Windows 7 machines, depicted as red, green, yellow and blue in the visualisation. We capture CPU usage, RAM usage, network protocol activity and system process activity, as well as the corresponding system screenshots that in this use case would notify the user of the ransomware infection. We utilise the default parameters for capturing data and visualising the activity, as described previously.

WannaCry

In our first experiment, we use the WannaCry ransomware. This particular malware was chosen because of its notoriety for using many resources upon execution and its ability to propagate across insecure machines extremely quickly.

Due to the wide coverage gained by the WannaCry outbreak in 2017, we are able to conduct an initial investigation based on existing technical analysis reports, such as that by Vipre Labs [17]. After WannaCry is initially executed it first checks for the presence of a specific website. If this domain is down then the attack continues. This domain was famously purchased and sink-holed, which was how the outbreak was originally contained. Given we are experimenting using an internal only network, the sample is able to execute as no internet connection is made available. Next the sample begins the encryption phase, where it spawns itself as a separate service called MSSECSVC2.0 which then drops and executes a 3MB Win32 portable executable file called tasksche.exe. This newly dropped executable is the ransomware component which performs file encryption using RSA–AES.

After the ransomware component exits, the MSSECSVC2.0 process remains running, now beginning to prepare for its propagation. It does this by creating two threads, one for LAN and one for WAN. In the LAN thread, it takes the first three octets of every IPv4 address associated with network adaptor and builds an IP list from 1 to 254 for the fourth octet. Each generated IP is passed into its own thread to execute the EternalBlue exploit, with a maximum of ten threads. In the WAN propagation thread, 128 public IP addresses are randomly generated, the first octet is pseudo-randomly generated, since it skips 127 and must be less then 224, the remaining octets are completely random. To propagate the thread connects to port 445 on the target machine, which is used by the Server Message Block (SMB) protocol. If the connection is successful it begins negotiating for the SMB tree ID, following this it makes five attempts to send a packet based on the EternalBlue exploit. Then it will be expecting a response from its target containing 0x51, which would mean the exploit has been successful and it can send the payload with the DoublePulsar shellcode. After this, the activity on the newly infected machine will proceed the same as the initial infection.

Given this technical analysis of the malware characteristics, we seek to explore how well this chain of events can be examined based on the network and system characteristics gathered by our data collector, and as visualised by our pixel-based approach.

Figure 4 shows the CPU, RAM and corresponding screenshots, as formulated by the visualisation tool. At the point of execution, the CPU usage of machine one can be seen to increase, and this increase in CPU usage is trigged across the other machines on the network despite no user interaction. This increase of activity would seem reasonable to correlate with the encryption process made by the machines once the ransomware component is in place.

Figure 5 shows the network usage for the WannaCry propagation. There is a clear pattern across all machines involving the increase in SMB activity as the dominant protocol in use, as we would expect from the known characteristics of the malware. We can see the communications between the machines, in particular, where red (machine one) is seen to then communicate with all other machines on the network, and likewise, they communicate back.

Figure 6 shows an extract of the process usage for the WannaCry propagation (the full visualisation is too large to depict, due to 40 rows per machine). The top row shows the execution of the malicious executable (file hash beginning 24d0). Subsequent processes can be observed including

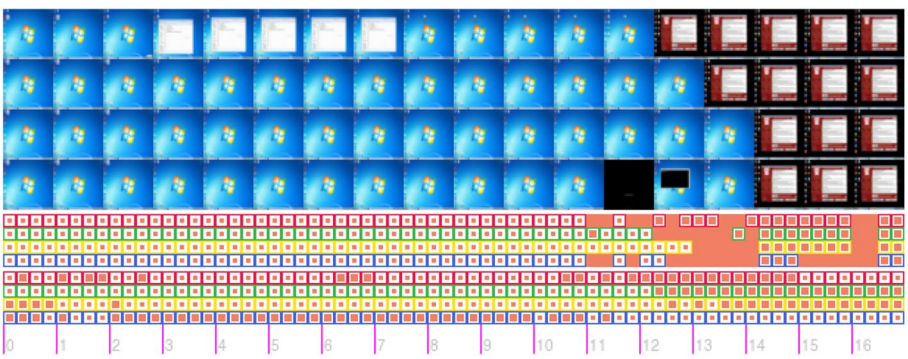

Fig. 4 WannaCry: Screen, CPU and RAM Usage

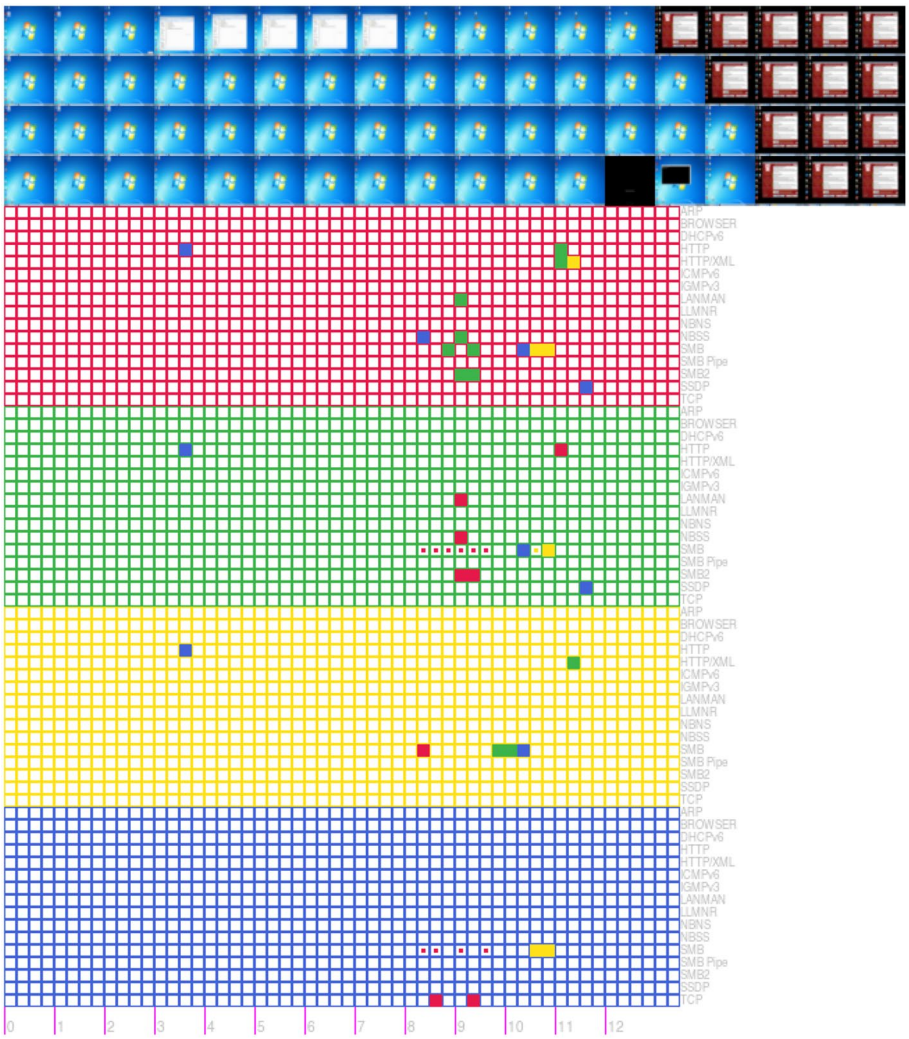

Fig. 5 WannaCry: Network Protocol Usage (where each row denotes an individual system-protocol name pair)

tasksche.exe, taskhsvc.exe, VSSVC.exe, WmiPrvSE.exe and @WanaDecryptor@.exe.

NotPetya

NotPetya is another form of ransomware which extended on the Petya malware but also incorporates the worm component of EternalBlue. As before, we can examine the technical analysis given by VMWare Carbon Black [3] to understand the expected behaviours.

Upon execution the malware first performs a check to assess what privilege level it is running as, primarily checking for the SeDebugPrivilege permission. Based on this permission the malware proceeds in one of two ways: if it does have the privilege then the malware assumes it has administrator privileges and encrypts the Master Boot Record (MBR) using the known Petya method; if the user does not have administrative privilege, the malware will instead use a user space encryption routine. The next stage of the process is to check for the presence of processes relating to anti-virus software. If any AV software is found then the global variable are modified to disarm these. If no processes are found and the malware has administrator privileges, it runs a credential stealer. It will then attempt to propagate using the EternalBlue SMB propagation as described earlier. After these routines complete, the malware will attempt to remove forensic artefact data by running a command to clear logs for Setup, System, Security, and Application, it will then delete the Update Sequence Number (USN) journal from the drive. Following this, the malware will force a restart in one of three ways depending on the privilege level. If the malware has access to SeShutDown and SeDebug, it will call the InitiateSystemShutdown API; if it only has access to SeDebug, it will initiate shutdown by calling the NtDll.NtRaiseHardError API; if it has neither, then it will rely on a created scheduled restart task. Since the MBR is already compromised, the malware boots to its own environment, where it then encrypts the Master File Table (MFT) and alerts the user to the compromise.

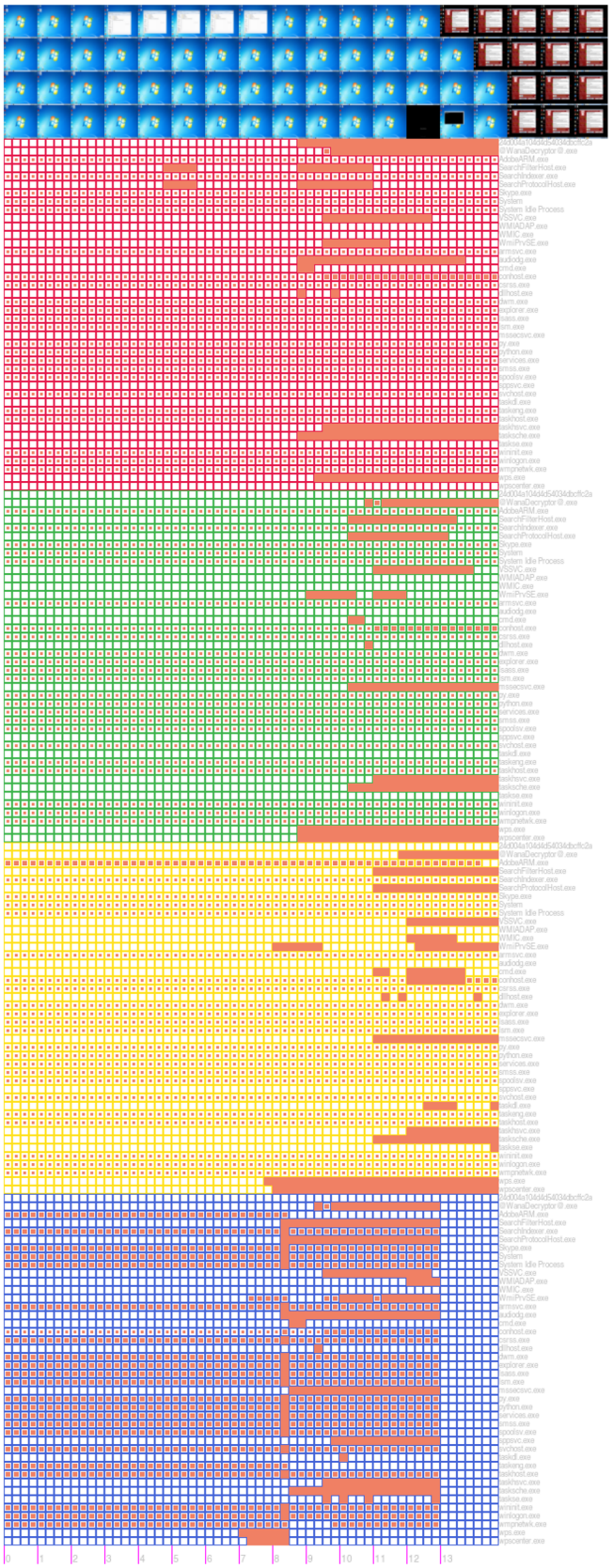

Fig. 6 WannaCry: System Process Usage (where each row denotes an individual system-process name pair)

Using this knowledge, we assumed that the activity would begin with a short spike, followed by a plateau of higher system resource usage. Following these would be an increase in SMB activity across the network, which would then be succeeded by another spike in system resource usage to denote the restart and encryption of the MFT.

Figure 7 shows the network activity observed when the NotPetya sample is executed on machine one in our virtualised network. First, we note the significant longer period of time for malware execution required here, since NotPetya will delay the restart process by 1 h to maximise the propagation stage before performing the system restart and the encryption process. Our visualisation approach scales well to handle a significantly longer period of time while still being able to convey key details. In an operational environment, analysts would naturally zoom and pan across the visualisation, to gain both an overview (as shown) as well as zooming in for a detailed view. The network analysis also reveals reoccurring patterns across all four machines, where the ICMP, LANMAN, NBSS, SMB and SMB2 protocols are all triggered. The pixel-based visualisation essentially creates a form of visual signature that could be used by analysts to identify related occurrences of protocols across systems.

Figure 8 shows the system processes that are called when the NotPetya sample is executed. Given the increased runtime of the malware before the system restarts, we have a signficantly longer period of time that requires analysis. The pixel visualisation allows us to identify distinct patterns across machines, for example, all machines call VSSVC.exe and WmiPrvSE.exe within the same time period. The process rundll32 is also triggered as a new activity across all compromised machines.

Discussion

We now reflect on the key findings from our investigation. Pixel-based visualisations are well suited to convey large volumes of data within a concise visual format, and are, therefore, well-suited for the security domain. While alternative approaches such as horizon charts also seek to compress information in a smaller visual space, for encoding additional data attributes such as network communication and destination machines, as well as for the identification of recurring patterns, we found pixel-based approaches provide greater flexibility such as being able to extend visual attributes, such as colour, size, and position, to accommodate for more complex data sources.

Pixel-based visualisation has the benefit of being able to display much more information in a confined space. If one was to visualise the entire network activity of our setup in a multi-line time-series plot, it would become cluttered extremely quickly. Moreover, it would be difficult to make associations and identify the relationship between data points. With the pixel-based approach, we avoid issues of data occlusion, and so any patterns that may be present are visible. Due to the compact representation, being able to illustrate multiple rows to further separate out data, such as network protocol usage, can be achieved still within a constrained visual area. Furthermore, using colour and size, we can denote additional data attributes, such as destination machine, and the volume of activity. This reveals clusters of activity in a much clearer fashion that a line plot. A traditional line plot may introduce much redundancy in terms of white space, whereby the security analyst may only be concerned in whether an attribute is within some

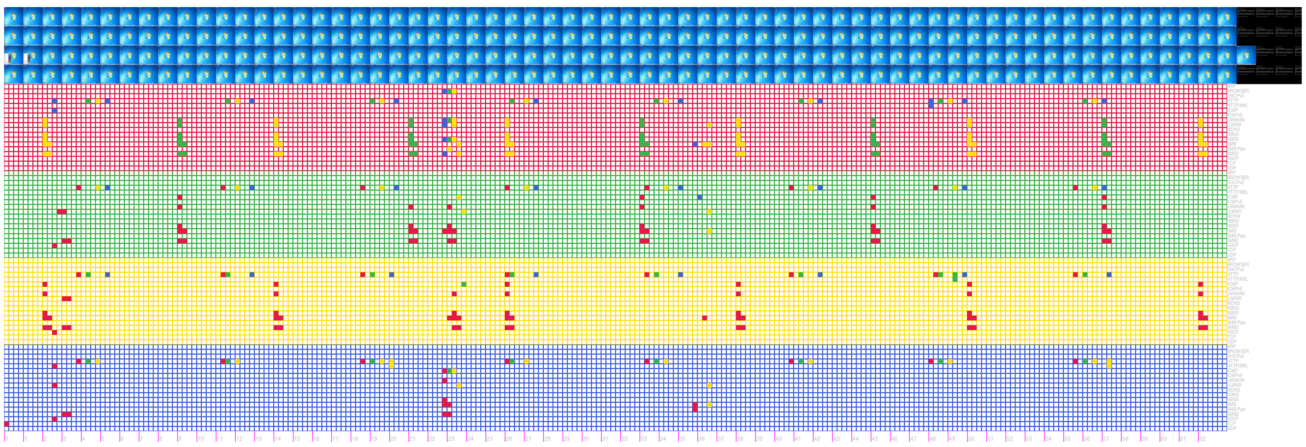

Fig. 7 NotPetya Network Protocol Analysis: Each row represents a protocol, grouped by machine (colour). Markers are scaled by amount of traffic (5-point scale), and coloured by destination machine. Over the 1 h period before NotPetya forces a system reboot, the visualisation reveals the protocol pattern. There is a recurring pattern of activity based on ICMP, LANMAN, NBSS, SMB, and SMB2 protocols that can be observed

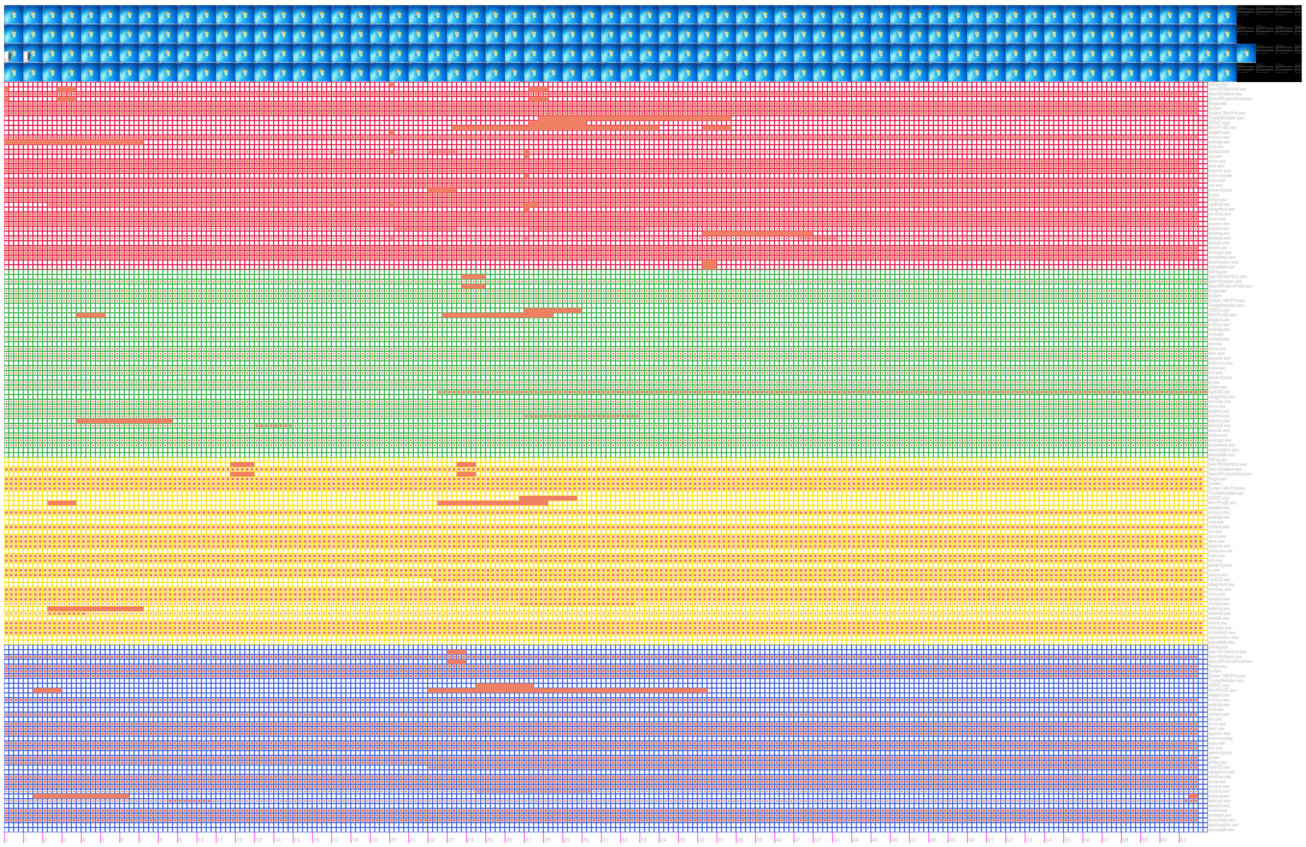

Fig. 8 NotPetya System Process Analysis: Each row represents a process call, grouped by machine (colour), using a 5-point scale for pixel size. Even with a highly compact view due to the increased runtime before malicious action, we can recognise common patterns of process calls across machines, including the increased activity to VSSVC.exe and WmiPrvSE.exe, and the start of the rundll32 process across each machine

threshold group (e.g., high, medium, low, none) or when there is change between these groups. Using a 9×9 pixel region provides up to 6 distinct thresholds which we found sufficient for representing system and network variations to observe system activities.

We were also able to visualise activity relating to processes running on each node, gathered by our Python script agent utilising the psutil library. Using the data gathered by this agent we could observe activity specifically related to the sample and compare it against a baseline of other processes present on the node. This is useful, because samples often create different processes to perform different tasks, by gathering this specific data we are able to see all of these and note when the process started, how many resources it used, and when it ceased. A good example of this is in Fig. 6, where you can see the different times that @WanaDecryptor@.exe executed on each machine, which could inform our hypotheses for the order in which the machines were infected. The data collector actually provides additional information that is not utilised fully yet, such as which process initiates which network communications, which could be incorporated as part of future work. One challenge in any sandbox monitoring agent is whether the agent itself is compromised or interferes with the malware execution. Many malware samples look for environment parameters that may suggest they are running in a sandbox [19], such as the present of *agent.py* which is used by Cuckoo or the phrase "VMware" occurring in the Windows registry. We found that our various data collectors would terminate at different times, which is likely related to this. Our Python script stops returning data just before the encryption process is complete, whereas the CPU and RAM usage is continually gathered as these are retrieved through the VBoxManage *getmetrics* command. It is suspected that the agent script fails due to the execution of WannaCry, causing the script to stop prematurely, because it cannot access resources as the encryption and EternalBlue routines are deployed.

Given the features of pixel-based visualisations, we believe such an approach would be valuable in setting, where fast recognition of propagation patterns would be key. For

example, SOC analysts are required to triage incidents of malware infections quickly to minimise potential damage to the network environment. If the traffic occurring on the network could be captured, visualised, and then compared against other previously captured samples, it may be possible to observe similarities between the activities and triage the incidents with greater accuracy. This can even apply to large amounts of data, since pixel-oriented methods allow for the condensing or broadening of data without suffering a loss of accuracy. This can been observed amongst our two samples, our WannaCry testing took place over 15 min and our NotPetya testing took place over an hour, yet despite this both tests created a suite of visualisations that can be used to discover patterns of activity. Future research will study how significantly larger networks could be examined using similar techniques in conjunction with other forms of data analysis and filtering, such as machine learning methods. Given the use of well-established industry tools, such as Wireshark, a further research question would explore how such novel methods are integrated with the tools, techniques and workflows of analysts to support both overview and detail as part of a triage process.

Conclusion

In this paper we have explored visualisation techniques for examining system and network-based attributes, utilising pixel-based visualisation to achieve an informative yet scalable view of a large corporate network. We use this to conduct malware propagation analysis, observing the protocol and process usage across networked machines for distributing and executing ransomware. While we use two malware variants that are now patched in modern systems, the principles remain crucial to have such network and system visualisation methods available for SOC analysts when tasked with new vulnerabilities.

Future work will investigate pro-active defenses against malware, environment-sensitive malware analysis, refinement of analysis functionality to enable specific views of individual machines and their metrics, and how visualisation techniques can be integrated for large-scale network-wide monitoring and assessment to help SOC analysts better identify and defend against potential attacks.

Author Contributions JW led the research activity under the supervision of PL. Both contributed to the conceptualisation, design, implementation, and experimentation. Both authors contributed to the production of the final manuscript.

Funding We would like to thank the UK National Cyber Security Centre studentship scheme for supporting this research.

Availability of Data and Material All software code and data related to this research is available at: https://github.com/phillegg/pixsys.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Afianian A, Niksefat S, Sadeghiyan B, Baptiste D. Malware dynamic analysis evasion techniques: a survey. ACM Comput Surv. 2019. https://doi.org/10.1145/3365001.
2. Bai J, Wang J, Zou G. A malware detection scheme based on mining format information. Sci World J. 2014. https://doi.org/10.1155/2014/260905.
3. Black C. Carbon black threat research technical analysis: Petya / notpetya ransomware. VMWare, Palo Alto, California: Tech. rep; 2018.
4. Borgo R, Proctor K, Chen M, Jänicke H, Murray T, Thornton I. Evaluating the impact of task demands and block resolution on the effectiveness of pixel-based visualization. IEEE Trans Vis Comput Graph. 2010;16(6):963–72. https://doi.org/10.1109/TVCG.2010.150.
5. Borkin MA, Vo AA, Bylinskii Z, Isola P, Sunkavalli S, Oliva A, Pfister H. What makes a visualization memorable? IEEE Trans Vis Comput Graph. 2013;19(12):2306–15. https://doi.org/10.1109/TVCG.2013.234.
6. Chakkaravarthy S, Sangeetha D, Vaidehi V. A survey on malware analysis and mitigation techniques. Comput Sci Rev. 2019;32:1–23. https://doi.org/10.1016/j.cosrev.2019.01.002.
7. Chen Q, Bridges RA. Automated behavioral analysis of malware: a case study of wannacry ransomware. In: 2017 16th IEEE International Conference on machine learning and applications (ICMLA), 2017; p. 454–60. https://doi.org/10.1109/ICMLA.2017.0-119.
8. Creese S, Goldsmith M, Moffat N, Happa J, Agrafiotis I. Cybervis: visualizing the potential impact of cyber attacks on the wider enterprise. In 2013; p. 73–9. https://doi.org/10.1109/THS.2013.6698979.
9. Donahue J, Paturi A, Mukkamala S. Visualization techniques for efficient malware detection. In: 2013; p. 289–91. https://doi.org/10.1109/ISI.2013.6578845.
10. Gove R, Deason L. Visualizing automatically detected periodic network activity. In: 2018; p. 1–8. https://doi.org/10.1109/VIZSEC.2018.8709177.
11. Grégio A, Santos R. Visualization techniques for malware behavior analysis. In: Proceedings of SPIE—the International

Society for Optical Engineering. 2011. https://doi.org/10.1117/12.883441.
12. Guillén JDH, Rey ÁM. Modeling malware propagation using a carrier compartment. Commun Nonlinear Sci Numer Simul. 2018;56:217–26.
13. Han K, Kang B, Im EG. Malware analysis using visualized image matrices. Sci World J. 2014. https://doi.org/10.1155/2014/132713.
14. Hosseini S, Abdollahi Azgomi M. A model for malware propagation in scale-free networks based on rumor spreading process. Comput Netw. 2016. https://doi.org/10.1016/j.comnet.2016.08.010.
15. Jamalpur S, Navya YS, Raja P, Tagore G, Rao GRK. Dynamic malware analysis using cuckoo sandbox. In: 2018; p. 1056–60. https://doi.org/10.1109/ICICCT.2018.8473346.
16. Keim D. Designing pixel-oriented visualization techniques: theory and applications. IEEE Trans Vis Comput Graph. 2000;6(1):59–78. https://doi.org/10.1109/2945.841121.
17. Labs V. Wannacry technical analysis. VIPRE Labs, Los Angeles, California: Tech. rep; 2017.
18. Lindorfer M, Kolbitsch C, Milani Comparetti P. Detecting environment-sensitive malware. In: Sommer R, Balzarotti D, Maier G, editors. Recent advances in intrusion detection. Berlin: Springer; 2011. p. 338–57.
19. Mills A, Legg P. Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques. J Cybersecur Privacy. 2021;1(1):19–39. https://doi.org/10.3390/jcp1010003, https://www.mdpi.com/2624-800X/1/1/3.
20. Mills A, Spyridopoulos T, Legg P. Efficient and interpretable real-time malware detection using random-forest. In: 2019 International Conference on cyber situational awareness, data analytics and assessment (Cyber SA), 2019; p. 1–8. https://doi.org/10.1109/CyberSA.2019.8899533.
21. Miramirkhani N, Appini MP, Nikiforakis N, Polychronakis M. Spotless sandboxes: evading malware analysis systems using wear-and-tear artifacts. In: 2017; p. 1009–24. https://doi.org/10.1109/SP.2017.42.
22. Patel A, Tailor J. A malicious activity monitoring mechanism to detect and prevent ransomware. Comput Fraud Secur. 2020;2020:14–9. https://doi.org/10.1016/S1361-3723(20)30009-9.
23. Rhode M, Burnap P, Jones K. Early stage malware prediction using recurrent neural networks. Comput Secur. 2017. https://doi.org/10.1016/j.cose.2018.05.010.
24. Satheesh Kumar M, Ben-Othman J, Srinivasagan K. An investigation on wannacry ransomware and its detection. In: 2018; p. 1–6. https://doi.org/10.1109/ISCC.2018.8538354.
25. Sharafaldin I, Habibi Lashkari A, Ghorbani A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: 2018; p. 108–116. https://doi.org/10.5220/0006639801080116.
26. Wagner M, Fischer F, Luh R, Haberson A, Rind A, Keim D.A, Aigner W. A survey of visualization systems for malware analysis. In: Borgo R, Ganovelli F, Viola I editors. Eurographics Conference on Visualization (EuroVis)—STARs. The Eurographics Association 2015. https://doi.org/10.2312/eurovisstar.20151114.
27. Xiao F, Lin Z, Sun Y, Ma Y. Malware detection based on deep learning of behavior graphs. Math Probl Eng. 2019;2019:1–10. https://doi.org/10.1155/2019/8195395.
28. Yu S, Gu G, Barnawi A, Guo S, Stojmenovic I. Malware propagation in large-scale networks. IEEE Trans Knowl Data Eng. 2015;27:170–9. https://doi.org/10.1109/TKDE.2014.2320725.
29. Zhuo W, Nadjin Y. Malwarevis: entity-based visualization of malware network traces. In: 2012; p. 41–47. https://doi.org/10.1145/2379690.2379696