University of the West of England

# Can Visualizing Malware Propagation Across Local Area Networks Assist in Security Decision Making?

Jacob John Williams

Cyber Security MSc

"This study was completed for the MA/MSc in Cyber Security at the University of the West of England, Bristol. The work is my own. Where the work of others is used or drawn on it is attributed".

Word Count: 13,566

# 1 Abstract

Malware is a forever escalating threat, the eternal game of cat and mouse between malware writers and cyber security professionals. Recent years has seen a dramatic rise in cases of businesses and institutions being infected and the resultant chaos costing copious amounts of money and affecting countless peoples lives. Therefore security practices have adapted to prevent such disasters, methods such as honeypots have become commonplace, and as a result there needs to be data to assist in the decision making behind such techniques. This paper details a quantitative study of various malware samples; gathering system metric data and network log data in order to visualize it and then determine if said visualizations can be used to assist in said decision making and therefore construct more secure network. To perform these tasks, this study also oversees the creation of a dashboard tool that can be used to launch virtual machines, create a virtual network, and gather the system metric data from outside the environment.

# 2 Introduction

Malware is perhaps one of the greatest threats in modern day IT, it is constantly evolving both terms of who it targets and what it exploits. Due to the evolutionary nature of malware modern security has turned into a game of cat and mouse, security analysts and malware writers always vying to overtake the other by finding new ways to protect or exploit systems. Unfortunately it is common in the industry for the malware writers always to be numerous steps ahead, with those who work in security being forced to operate reactively to malware, this is why the security industry needs more who are willing to study malware in order to prevent it.

How malware propagates is key to understand how malware operates, external propagation; that being how malware bridges the air gap between networks is commonly studied and is fairly established in various forms of study, mathematically and visually. How malware propagates within a local area network however is studied rather one sidedly, leaning towards the mathematical side. These studies are extremely comprehensive and explore numerous methodologies of formulating how malware propagates, but it requires some complex knowledge and understanding of mathematical models and theories and as a result could be seen as being rather unfriendly to those wishing to involve themselves with Cyber Security from other backgrounds. Therefore the goal of this study to to attempt to visualise malware propagation using variables and characteristics from machines on local area networks and produce a more easily interpretable way of showing the nature of propagation in order to assist in security decision making.

## 2.1 Aims and Objectives

1. To create clear and interpretable visualisations of propagations.
    a) To inform other researchers of the details of the propagation in order to assist in the further development of anti-malware capabilities.
    b) To inform those in positions whose decisions can affect the security of a network, and where they should be deploying network security features.
    c) To aid students in the study of malware behaviour and its impact across networks.
2. To identify key characteristics of malware propagation that can be used in the response to an attack.
    a) Is malware propagation in certain malware deterministic?
    b) Does IP address locality affect the nature of propagation?
    c) How does network structure affect propagation?
    d) Can this be used to effectively respond to an active threat event?

The first objective can be considered a general committal, the effort to expand the knowledge surrounding the nature of malware. One must acknowledge that we will be studying existing malware and are not necessarily looking to find brand new characteristics, but rather create visualizations that better display the nature of malware propagation.

The second objective is the crux of this research, an attempt to create better response times to an active attack event. This paper will explore whether what is gathered from the tool can be used to assist in decision making, such as dynamically disconnecting clusters of or

singular machines from the network in an attempt to protect the greater network, or perhaps in a more complex manner, assist in the dynamic creation of HoneyPots or HoneyNets in order to protect networks or observe active malware events.

## 3 Related Literature

As aforementioned, the vast majority of research into malware propagation is mathematical in nature, but this does not make it irrelevant to this study, as researching what we already know of malware propagation will allow us to create hypotheses and possibly support not only our own findings but others findings as well. *Yu et al (2015)* explored the prospect of malware propagation by applying epidemic theory and creating a two-layer epidemic model. Yu et al (2015) focus on epidemic modelling because it is *"more focused on the number of compromised hosts and their distributions"* compared to another method called Control System Theory, which Yu et al (2015) claims are more inclined to *"try to detect and contain the spread of malware"*. This distinction is interesting as it categorises the two types of studies seen in the field of malware propagation, those who observe the spread of malware to define its nature are studying it Epidemiologically, and those study it to configure systems to prevent propagation are studying it under Control System Theory. With this distinction in mind, one could claim that this study will be observing Epidemiologically. Yu et als (2015) results are interesting, providing evidence that propagation in large scale networks possess three different discernible stages, *"Exponential distribution in its early stage"*, *"power law distribution with short tail in late stage"*, and *"power law distribution in its final stage"*. This aligns with how one might imagine malware propagation, with it at first infecting everything it can and once there is nothing more to infect the rate of infection quite obviously decreases exceedingly quickly. Yu et al (2015) explores the rate at which malware propagates to great depths, but like many mathematical models doesn't explore the nature of propagation, such as whether the way it happens is deterministic or if IP address locality matters.

Guillen and Rey (2018) researched what is in concept an extension of Yu et als (2015) work. Guillen and Rey (2018) acknowledge the existence of *"compartment devices"* which are devices that *"cannot be directly targeted by malware, but can be used to propagate it"*. It presents an interesting figurative, the devices that propagate without being infected do not contribute towards the infection rate themselves but they greatly increase the general infection rate. This presented an interesting line of though, that if you were to visualise malware propagation solely through signatures or system resources then these compartment devices would seemingly be unaffected but devices near to them would seemingly spontaneously become infected. If one was to attempt to study malware behaviour in networks that frequently possess these compartment devices then network trace analysis would be key. Hosseni and Azgomi (2016) formulated a model for representing malware propagation based on a rumour spreading model. Identifying five different types of machines (nodes) that perform different roles in the propagation of malware, these being *"susceptible, exposed, infectious, recovered, vaccinated"*. This study lends well to classifying the different states a machine in an active attack on a LAN may take and may be useful to consider during visualization.

Zhuo and Nadjin (2012) conducted a study with the goal of visualizing malware network traces, in doing so they created the tool MalwareVis. MalwareVis is able to pull heterogeneous attributes like protocol, IP address, and more from the network trace log. Network traces are a key part of categorizing malware since often the malware has to communicate with an outside command and control server in order to deploy, receive orders, or exfiltrate data. Analysing network traces is often a long and arduous job in malware analysis because the logs contain large volumes of data, as a result, the study focuses more on specific protocols such as TCP and DNS. Whilst the study is interesting and attempts to visualize a typically complex part of malware analysis, it still produces a visualisation that cannot be understood without in-depth study of the formulas it is built upon. To the average researcher this will prove no trouble, but perhaps to those getting into the field it will prove much more difficult and perhaps yield very little to their understanding. Gove and Deason (2018) also explore the concept of identifying malware through network activity. The target of their study is the period network activity of malware, which they acknowledge as being a difficult task due to it being *"easily drowned out by non-malicious network activity"*. Gove

and Deason (2018) utilise a novel algorithm based on Discrete Fourier Transforms, and they pair the output of this with aggregation summary tables that will inform users which detection are worth investigating and which are not. This study is highly complex but also very interesting, as it presents a method of analysing data that is usually very difficult to obtain. The downside is the method in which the data is collected is similarly as complex, whilst network traces might be an excellent way to study propagation in particular it is appearing that such a method may not be conceivable within the time frame of our study.

The study of malware detection is useful to this study due to the key overlap of malware characteristics. If certain characteristics are better at identifying malware then they will most certainly be better for visualising malware, therefore much study was conducted into the detection of malware and the identification of their characteristics. Rhode et al (2018) conducted study where they used Recurrent Neural Networks to predict malware, in the way of investigating *"whether or not an executable is malicious based on a short snapshot of behavioural data"*. The recurrent neural network works using hyper-parameters, but what these hyper-parameters work with is what is interesting. Rhode et al (2018) compile minimum and maximum values of inputs to identify the benign or malicious nature of samples. These input values are things such as *"total processes, max process ID, cpu user (%), cpu system (%)"* and more, in short they identify malicious activity when the executable exhibits resource usage past a certain threshold which is typical of other malicious samples. This study is useful as it presents the concept of using system resources and set thresholds to visualise malware, one could visualise all resource usage but have the visualisation be different for those who pass the said threshold, perhaps in a green-amber-red fashion.

Mills et al also used a form of machine learning in order to detect malware. Mills et al used Random Forest to create NODENS, a lightweight malware detection platform that can be deployed on cheaper hardware. After *"removing duplicate or unnecessary attributes"* a total of twenty-two features were identified for the classification model. All these characteristics were once again related to system resource usage, such as *"total processor time, user processor time, Non-paged / paged memory sizes"* and more. This reinforces the proposed method of visualising propagation, since resource usage is a recurring method of signalling what is and is not malware. Xiao et al (2019) conducted a study with the goal of detecting malware through behaviour graphs. The behaviour graphs were constructed by monitoring and gathering the order in which malware uses system calls, with this you can deem certain orders of system calls benign and others malicious. This study is interesting as well as comprehensive, and provides a good amount of detail in regards to what systems calls malware frequently uses that one could use to get started conducting a similar study. Our study will probably not utilise such a method, but it is worth noting that the system calls are usually in relation to a system resource such as CPU, memory, etc.

Bai et al (2014) used a method of mining format information from malware executables in order to identify characteristics that would make an executable appear malicious. They identified 197 different features and used them to train an algorithm that was able to achieve 99.1% accuracy in the testing environment. Whilst this method of detection is interesting and certainly seems effective, the characteristics it identifies assist in detecting malware before it is able to execute, which is not particularly useful to us when we need the malware to execute and propagate. What is useful however is that some of the characteristics captured seem to relate in some way to how the executable will go about using system resources when executed.

Patel and Tailor (2020) researched a method of not only monitoring a system for ransomware, but also counteracting it. To counteract the ransomware, the system they have produced first identifies the encryption of a protected folder it is explicitly monitoring, once it has it creates a large dummy file for the ransomware to encrypt. Whilst it is spending the time encrypting the file, the system changes its properties to read only so that the ransomware can no longer encrypt the files on the machine. Whilst what they've created is interesting and somewhat unique, one must question the usefulness of it since it doesn't particularly prevent ransomware, but merely gives you an opportunity to prevent it. One must also question the validity of the testing, since the study full well admits that *"we designed the attacks"* and one could argue due to this it doesn't necessarily reflect a real world scenario. Unlike the others, the method this study uses to identify ransomware isn't particularly useful, as the reliability of identifying ransomware based on a monitored folder may prove unreliable when extracting data since the functions used to extract said data may

already be blocked by the ransomware before it is counteracted or the preventative measure of locking down the system may create a scenario where we cannot extract the data.

It was also deemed worthwhile to comprehensively research malware analysis methods, particularly in regards to sandboxes, since they will be used to observe the propagation in the first place. The first paper by Afianian et al (2019) is a comprehensive summary of knowledge regarding common methods that malware use to evade detection and analysis. The paper begins begins by acknowledging that modern day defensive and analysis techniques can be *"readily foiled by zero-day fingerprinting techniques or other evasion tactics such as stalling"*. The paper then walks through the different studied methods used to avoid analysis, dividing them into two distinct categories: detection dependent evasion, where the *"goal is to detect its environment to verify if it is a sandbox or not"*; and detection independent evasion, which are methods that do *"not rely on detecting the environment"*. This paper as mentioned previously is very comprehensive and therefore very useful for learning how malware avoids sandboxing in a very brief and concise manner, even giving explicit examples of what malware tends to look for in regards to environmental details. Chakkaravarthy et al (2019) take a step back from individual machines and take a look at methods malware uses to avoid detection within the network, such as payload fragmentation, session splicing, and more. All of these details whilst interesting were not our key focus on this report, what was is the fact that the paper includes full details on the configuration of their virtual machines including OS, network interface, processor/core/memory, etc. Interestingly enough, this study seems to use NAT based configuration which would mean the infected virtual network still has connection to the host machine, presenting a possible danger.

Miramirkhani et al (2017) also explored the creation of sandbox machines for observing malware and took a deep dive into what small features in a virtual machine might be used by malware writers to detect a virtual environment. Miramirkhani et al (2017) identified different *"Artifacts",* naming *"Direct Artifacts"* which are artifacts created by a users direct interaction, and *"Indirect Artifacts"* which are created by general system activity. Miramirkhani et al (2017) focused only on Indirect Artifacts, stating that whilst Direct Artifacts are qualitatively stronger, they also pose an issue of privacy to investigate, since their research would require the use of participants on their own machines. Miramirkhani et al (2017) identified multiple subcategories of Artifacts, often relating to machine functionality, such as *"System, Disk, Network, Registry, and Browser".* Using these subcategories, Miramirkhani et al (2017) identified and organized their Artifacts, identifying many different metrics that differ greatly between the average user system and a Sandbox environment. Whilst it isnt in our projects aims to associate for context aware malware, it will be useful to explore at a surface level to create the most effective sandbox environment we can to consequently create accurate visualizations.

Sharafaldin et al (2017) explored the creation of a dataset for use with Intrusion Detection Systems. The dataset is completely labeled and contains more than 80 extracted network traffic features for the use in identifying benign or intrusive flows, to gather these they utilized CICFlowMeter, a publicly available software from the Canadian Institute for Cyber Security. The testing of their datasets was fairly comprehensive and consisted of the creation of two network types, the Victim and Attack networks. Sharafaldin et al (2017) further analyzed the dataset to designate different signatures to varying "Attack Profiles" which consist of attacks such as "Brute Force, Heartbleed, Botnet" and so on. This research will be useful if the tool created and utilized is going to be adapted to detect and visualize physical intrusion through an active attacker rather than just an autonomous malware, and is therefore worth keeping in mind, but such plans may be limited by the timed conditions of this project.

Creese et al (2013) undertook a project similar to our own, and attempted to visualize enterprise network attacks and their subsequent potential consequences. The visualization program utilizes traditional network diagram icons and combines them with business process modelling and notation, a disk propagation logic that connects the network and business process and task layer. Creese et als (2013) research is not only important to our project in their results, but also their methodology, since they acknowledge some key principles before and during the creation of their tool regarding exactly what their tool must possess. These design principles will surely be fitted into our own project. In terms of results, Creese et als (2013) work *"demonstrates that it is possible to determine and visualize the potential impact of a cyber attack",* using this sort of visualization technique we can create systems

that allow for greater clarity in the communication of risk not only to the technicians maintaining the networks we work on, but also to the decision makers than fund them.

# 4 Methodology

With the objectives in mind the approach to fulfilling them will be linear in nature. There is a defined order to which the objectives must be completed, not out of preference but out of dependency. Because of this how the project will progress is fairly predictable, the following sub section will provide detail and justifications for this.

## 4.1 Timeline

The first stage of this project will be the creation of the tool that allows us to extract details from an infected machine to later analyse the propagation. This tool must not only extract the various system details but also potentially extract the time in which the system appears to be infected. There are numerous subsections to this stage, both pre and post tool creation. Before we can begin creating the tool we must first research malware analysis methods concerning the creation of sandbox environments in order to assure that the system we are testing on will not allow the malware to propagate outside of the virtual environment. Furthermore, the methods that malware analysts use to trick malware with a weaker context awareness algorithms into thinking it is in fact in a legitimate users system must be explored. On the other side of the aisle, it will be prudent to research which malware exploit vulnerabilities that allow them to escape virtual environments, at first to avoid them, but if time allows possibly to study them specifically.

The creation of the tool will follow, and then be followed itself by the testing phases. At first the tool will be tested on systems that are not under active attack, then proceed to active attack testing once the extraction functionality is deemed acceptable. This section aligns with the first objective.

The next stage will involve the deployment of the tool to extract data from machines that are experiencing an active attack. Development will technically be continuing during this stage as different malware tend to exhibit different characteristics and therefore the tool may have to be adapted to suit best what the current active attack is. This is in alignment with objective three, the for the sake of transparency this project will include all of the tested characteristics, whereas the ones visualised may only be the ones with significant results.

The final stage will be to analyse, visualise, and draw conclusions from the results, keeping to the requirements outline within objective two. Stage two and three will occur multiple times, each run analysing a different malware. How many times this runs is determined by the amount of time surplus.

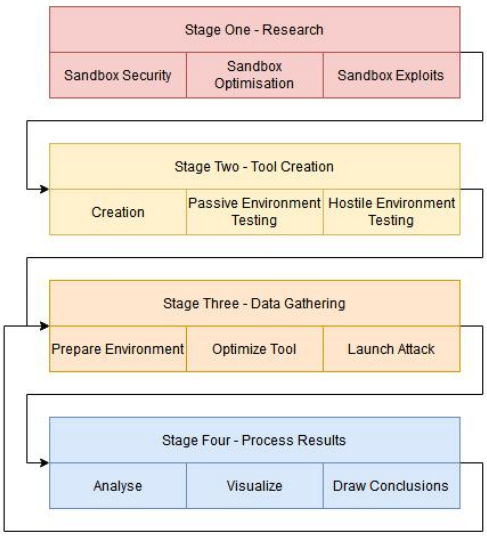


Figure 1 - Project Timeline

### 4.2 Ethics

There are two acknowledged ethical concerns around this project, one which acts as a requirement and another that must be declared. The requirement being that due to working with legitimate malware strains, the sandbox environment must be as secure as possible because anything less may allow the malware to propagate through the sandbox and into the home/work/school network, there is could pose a threat to any device on said network. Therefore one can consider it an ethical requirement to not only optimise the sandbox environment but also research the malware strains thoroughly for any potential context based exploits that may endanger the external network.

The declaration is that at no point in this project will malware be created, only malware that has already circulated, been studied, and prevented will be studied.

### 4.3 Testing Setup

In order to gather the appropriate data to visualize malware activity and propagation, we will be using a three tiered approach. The first two methods will be handled by a tool created especially for this project, which will gather metrics from all of the running virtual machines every five seconds and take screenshots of the current view every 30 seconds. This way we can view various possible changes occurring in the background via resource usage and we can also view obvious situation changes in the foreground of the runtime. For example, when running a WannaCry sample we would expect to see a sharp spike in resource usage at first and then the “ransom note” appear after it has finished its encryption operation, using these two methods we can observe both changes. The third method of gathering data will be utilizing WireShark to capture the communications between the machines on the testing network, WireShark will run on a separate node on the network which will be running in promiscuous mode and observing the primary network adaptor. This way we can view any and all activity occurring between the nodes.

In this paper the network setup will be fairly simplistic, being a LAN consisting of four vulnerable Windows 7 virtual machines. The machines will be configured to be as vulnerable as possible since our goal is to observe the way in which the malware propagates, to achieve this each virtual machine will have: an open, password-less, limited encryption fileshare on the network; all ports open; and disabled user access control. As mentioned prior, there will be a fifth node on the network running Ubuntu 20.04 LTS configured to network in promiscuous mode and it will only run WireShark. This network will be running entirely on Oracle Virtualbox’s internal network setup in order to prevent propagation from the virtual network setup into the testing environments host machine. The only machine that will have the ability to access the external host will be the Ubuntu machine, which will have a Virtualbox fileshare in order to extract the network activity csv. This fileshare will be disabled and ejected during active testing, and will only be mounted and reactivated after testing has concluded, the infected machines have been reverted to their clean snapshots, and the Ubuntu machine has been checked for signs of infection.

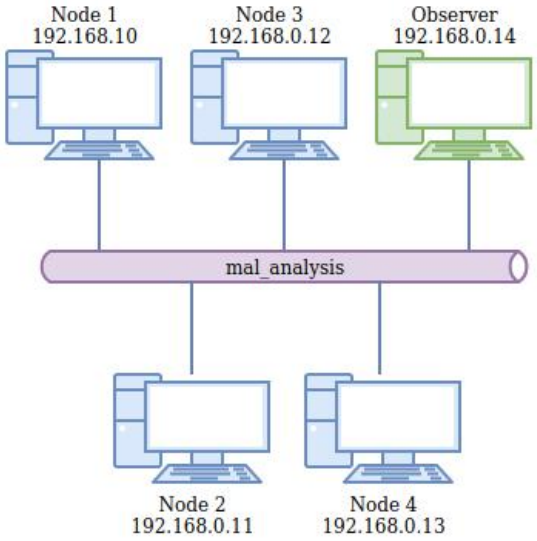

Figure 2 - Network Architecture

### 4.4 Testing Plan

The testing conducted will generally be quantitative in nature, the goal is to attempt to observe and visualize the activity and propagation of as many samples as possible in order to get a broad overview of the malware landscape and demonstrate the capabilities of our setup as much we can. This is not to say that the finding will be lacking detail however, as each sample will have the three different methods to analyse, and the network log will have many different aspects to be analyzed alone, such as the different protocols.The various samples being used will be acquired from the online malware archive TheZoo.

Each sample will be subject to the same amount of testing, twenty minutes in total: ten minutes of general activity, ten minutes of activity with the sample active starting on Node 1. Further time constraints will be setup and tear down time, which would take about ten minutes each, making the total amount of time per test around forty minutes given that there are no complications. This amount of time was decided upon to greater allow the use of a quantitative method, but also to standardize the amount of time we can analyse. It is entirely possible that some samples may not activate within the testing time, but this will be explored and noted if it has been determined to be typical or atypical

## 5 Testing and Visualization

Within this section we will conduct tests using various malware samples and utilise our aforementioned setup to gather appropriate metrics. The first subsection will contain a table briefly detailing every test conducted, the subsections following it will take a deeper look at the tests that appropriately showed propagation resulting in either a fully infected or partially infected network.

Our malware samples have been acquired from a few different sources, the first and probably the most notable is theZoo malware repository on Github, created by Yuval tisf Nativ and now maintained by Shahak Shalev. It is, for the most part, a community project intended to gather and catalogue different malware and different versions of said malware, community users can submit their own samples and have them entered where necessary. Another malware collection used is the MalwareDatabase maintained by a user called NTFS123. MalwareDatabase contains a few samples that are different to their theZoo counterparts, for example, the WannaCry sample in theZoo appears to be missing the EternalBlue worm component, whereas the WannaCry sample from MalwareDatabase does contain the EternalBlue worm component.

Alongside our created dashboard program, we utilized a few different tools in order to verify certain versions or characteristics of malware samples used. The first of these is a website called VirusTotal created by Hispasec Sistemas, this website allows you to analyse suspicious files or links and to view any previous entries with the same appearance. This can be done by submitting the file or URL, or by searching specifically for the hash values. VirusTotal was particularly useful in order to distinguish between the different versions of malware and how each sample is different to its previous, even if they appear the same. In the following section, the MD5 column can be entered into VirusTotal in order to view the full details of each sample we used. Another tool we used was the Cuckoo sandbox environment, an open source automated malware analysis environment. Cuckoo was used in this study as a backup to our own environment, for example, if we run a sample in our environment and it seemingly does nothing, but VirusTotal has multiple reports of it having visible activity, we would run it in the Cuckoo sandbox to see if the malware has some obvious contextual awareness capabilities.

### 5.1 Testing Overview

| Test no. | Sample | MD5 | Result | Expected(?) |
|---|---|---|---|---|
| 1 | WannaCry (Wormless Variant) | 84c82835a5d21bbcf75a61706d8ab549 | Infection of Node 1, No propagation. | Yes, general metric testing. |
| 2 | WannaCry | db349b97c37d22f5ea1d1841e3c | Propagated, network fully | Yes. |

| | | | | |
|---|---|---|---|---|
| | | 89eb4 | infected. | |
| 3 | CryptoLocker (November 2013) | 7f9c454a2e016e533e181d53eba113bc | Infection of Node 1, no encryption, no ransom, no propagation. | Mixed. Excepted no propagation. Didn't expected lack of encryption and ransom. Did infect machine and create startup processes, Cuckoo verified (Appendix 2). |
| 4 | CryptoLocker (September 2013) | 04fb36199787f2e3e2135611a38321eb | See (3). | See (3). |
| 5 | CryptoLocker (January 2014) | 829dde7015c32d7d77d8128665390dab / 0246bb54723bd4a49444aa4ca254845a | See (3). | See (3). |
| 6 | CryptoWall | 47363b94cee907e2b8926c1be61150c7 | No visible infection, no visible process, no encryption. | No. Behaved as one might expect of a Crypto-ransomware after being ran, but then effectively did nothing. |
| 7 | DirCrypt | N/A | N/A | N/A. TheZoo lists this as being Ransomware yet it doesnt appear to be a malware sample, but rather a program for decrypting encrypted files? |
| 8 | TeslaCrypt | 6d3d62a4cff19b4f2cc7ce9027c33be8 | Full infection, files encrypted. No propagation. | Yes. TeslaCrypt is not known to possess a propagation component. |
| 9 | TeslaCrypt | 6e080aa085293bb9fbdcc9015337d309 | Full infection, files encrypted. No propagation. | Yes. |
| 10 | TeslaCrypt | 209a288c68207d57e0ce6e60ebf60729 | Full infection, files encrypted. No propagation. | Yes. |
| 11 | Radamant | 6152709e741c4d5a5d793d35817b4c3d | Debatable infection, no visible encryption but computer | No. |

| | | | | |
|---|---|---|---|---|
| | | | becomes softlocked in WMI error. | |
| 12 | Radamant | b0625408735468e40f4af9472afcb35a | See (11). | N/A |
| 13 | Vipasana | 2aea3b217e6a3d08ef684594192cafc8 | Full infection, files encrypted, ransom note displayed. No propagation. | Yes. Vipasana is not known to possess a propagation component. |
| 14 | Vipasana | a890e2f924dea3cb3e46a95431ffae39 | See (13). | See (13). |
| 15 | Vipasana | adb5c262ca4f95fee36ae4b9b5d41d45 | See (13) | See (13). |
| 16 | Locky | b06d9dd17c69ed2ae75d9e40b2631b42 | Execution, self deletion, no encryption, no propagation. | Partially. Locky is known to be context aware and therefore may not have executed fully. |
| 17 | Petya | af2379cc4d607a45ac44d62135fb7015 | Execution, forced restart, encryption, ransomnote. No propagation. | Yes. Original Petya variant did not possess the EternalBlue exploit. |
| 18 | Petya | a92f13f3a1b3b39833d3cc336301b713 | See (17). | See (17). |
| 19 | NotPetya / PetrWrap | e5cc289b0b2b74b8e02f5a7f07867705 | Execution, no encryption or propagation, visible infection. Restart task. | No, expected propagation. |
| 20 | NotPetya / PetrWrap | 71b6a493388e7d0b40c83ce903bc6b04 | Execution, Self deletion, no encryption or propagation. Visible infection. Restart task. | No, Expected propagation. |
| 21 | NotPetya / PetrWrap | e5cc289b0b2b74b8e02f5a7f07867705 | Execution, propagation occurred, all machines infected. Took 1 hour and 10 minutes. | Yes, this was hypothesized. NotPetya acts in a delayed fashion. |
| 22 | Jigsaw | 2773e3dc59472296cb0024ba7715a64e | Execution, Encryption, ransomnote. No propagation. | Yes, Jigsaw is not known to propagate over LAN. |
| 23 | Satana | 46bfd4f1d581d7c0121d2b19a005d3df | Execution, generation of a txt ransomnote, no obvious | No, expected some encryption. Expected no |

| | | | | |
|---|---|---|---|---|
| | | | encryption. No propagation. | propagation. |
| 24 | Satana | 108756f41d114eb93e136ba2feb838d0 | See (23). | See (23). |
| 25 | Rex | 5bd44a35094fe6f7794d895122ddfa62 | Successful execution, no obvious infection, no obvious propagation. | Yes. |
| 26 | NSIS | 663fbf2a248971ea69c6234480a4bdcb | Execution failed. | No. |
| 27 | Bechiro | 0d06681f63f3026260aa1e15d86520a0 | Installation failed due to lack of internet connection. | No. |
| 28 | Artemis | caff801a280d42dbd1ad6b1266d3c43a | Installation completed, required “update”, connection failed. | No. |
| 29 | Kazy | ebefee9de7d429fe00593a1f6203cd6a | Executed, self deletion, no visible infection, no propagation. | Yes. |
| 30 | LoadMoney | d41d8cd98f00b204e9800998ecf8427e | Displayed corrupted alert, required confirmation, no activity after. | No. |
| 31 | Bladabindi | 5a559b6d223c79f3736dc52794636cfd | Executed, no visible infection, no visible propagation. | Yes. |
| 32 | Destrover | e904bf93403c0fb08b9683a9e858c73e | Executed, self deletion, no visible infection, no visible propagation. | Yes. |
| 33 | Adarmax | e33af9e602cbb7ac3634c2608150dd18 | Execution, no visible infection, no visible propagation. | Yes. |
| 34 | Stabuniq | f31b797831b36a4877aa0fd173a7a4a2 | Execution, self deletion, no visible infection, no visible propagation. | Yes. |
| 35 | ZeroAccess | fe756584b159fd24dc4b6a572917 | Dll loaded, no visible | Yes. |

| | | | | |
|---|---|---|---|---|
| | | 354c | infection, no visible propagation | |
| 36 | ZeroAccess | a2611095f689fadffd3068e0d4e3e7ed | Execution, no visible infection, no visible propagation. | Yes. |
| 37 | Emotet | 8baa9b809b591a11af423824f4d9726a | Execution, no visible infection, no visible propagation. | Partially, Emotet is known to be silent, Emotet is also a downloader its payload may not have been entirely deployed. |
| 38 | Rombertik | efc9040f587a5dd9e1de4707ec1ed8c5 | Execution, no visible execution, no visible propagation. | Yes. |
| 39 | Nitlove | b3962f61a4819593233aa5893421c4d1 | Execution, no visible infection, no visible propagation. | Yes. |
| 40 | Dino | ab2e178c77f6df518024a71d05e98451 | Execution, no visible infection, no visible propagation. | Yes. |
| 41 | Nivdort | ed2cd14a28ff2d00a5cefcf6a074af8d | Execution, visible infection, no visible propagation. | Yes. |
| 42 | Sality | d41d8cd98f00b204e9800998ecf8427e | Execution failed, missing prerequisite program. | No. |
| 43 | Beagle | b8a1c010e5cb3cf8c0a00a8e57e8c0b7 | No execution. | No. |
| 44 | Narilam | 8e63c306e95843eccab53dad31b3a98b | Questionable execution, loaded new spoofed file window, no other visible activity. | No. |
| 45 | Kovter | 15af6227d39ca3f9d1dcd8566efb0057 | Execution, self deletion, no visible infection, no visible propagation. | Yes. |
| 46 | Asprox | 53fec0c29fb30de88c9a6a369e8c | Execution, “error message” | Partially, did not expect the |

| | | | | |
|---|---|---|---|---|
| | | ae62 | displayed, no visible infection, no visible propagation. | fake error message. |
| 47 | Asprox | 0d655ecb0b27564685114e1d2e598627 | Dll loaded, fake error appeared roughly 3 minutes after the loading, much slower than (46). | No, usually dll based malwares work somewhat faster than their exe counterparts. |
| 48 | Asprox | d062d420e2ac73b0211afe30063807fa | See (46). | Yes. |
| 49 | Asprox | 400439a05c61cfd61625749a7f1fd8f1 | Execution, no fake error this variant, no visible infection, no visible propagation. | No, expected the fake error message like the other variants. |
| 50 | Mamba | 409d80bb94645fbc4a1fa61c07806883 | Execution, no visible infection, no visible propagation. | No. |

## 5.2 WannaCry [Wormless]

A wormless variant of WannaCry was used to fine tune the data gathering during development. This variant of WannaCry cannot propagate across a network like its famous sibling due to lacking the worm component of the payload that would allow WannaCry to exploit the EternalBlue vulnerability. This was used simply to reduce the tear down time of a test during development, but also to make sure that there was something to be recorded by the dashboard. WannaCry worked well for this job as it works fast and is very loud in terms of metric usage, meaning that its activity is fairly easy to spot and pin down at a particular point in time.

Many different tests were conducted and many changes made, the first tested included only the four nodes and the dashboard program, whereas the final test would also include the Ubuntu node to allow for the capture of network activity. This is the test we visualise below. We decided to visualise a test with only one infected machine for clarity as to what exactly activity on an infected machine looks like compared against machines that are not infected. Furthermore, the later tests which will show all network activity originating from and heading to multiple infected machines will get incredibly loud, and therefore we saw fit to include a test run where one can get a general impression of what our network visualisations will look like and how they differ to the later ones.

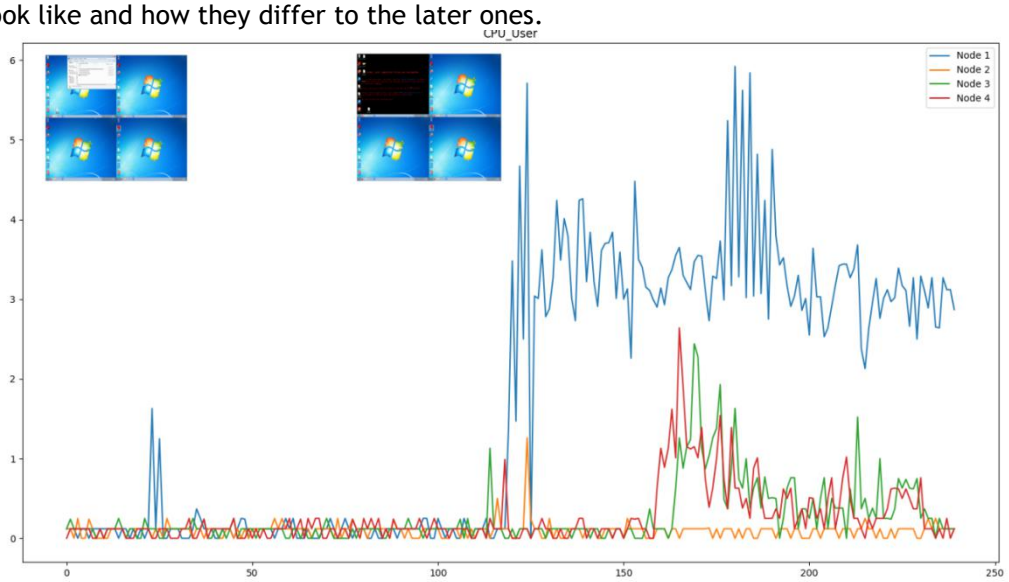


Figure 3 - WannaCry Wormless CPU

As you can see in Figure 3, it is extremely evident when the machine becomes infected and then encrypts data. This increased level of activity then continues throughout the remainder of the run time. Nodes 3 and 4 also see a strange spike in activity, the reason for this is unknown, as it would be impossible for them to become infected because the sample doesnt have the ability to propagate, therefore all we can conclude is that this increased activity might originate from a simultaneous bout of network activity of some kind, but in any way it is erroneous.

Interestingly, despite there being no worm component to this sample and therefore no EternalBlue consequently meaning there is no SMB exploit, the infected Node 1 still makes an attempt at creating an SMB connection with Node 2 as seen in Figure 4. This would imply that despite having no worm component, the sample still possesses the functionality that allows it to search for the SMB vulnerability and establish the connection, but not exploit it. This could mean that this particular WannaCry sample is either a neutered sample, its had the worm component removed, or it is a sample from a point in time when the worm component was still being implemented. Despite how outlandish it sounds, we believe the former to be more likely, since it seems strange for a malware author to release a malware that is faulty, which in itself isnt uncommon, but just unlikely.

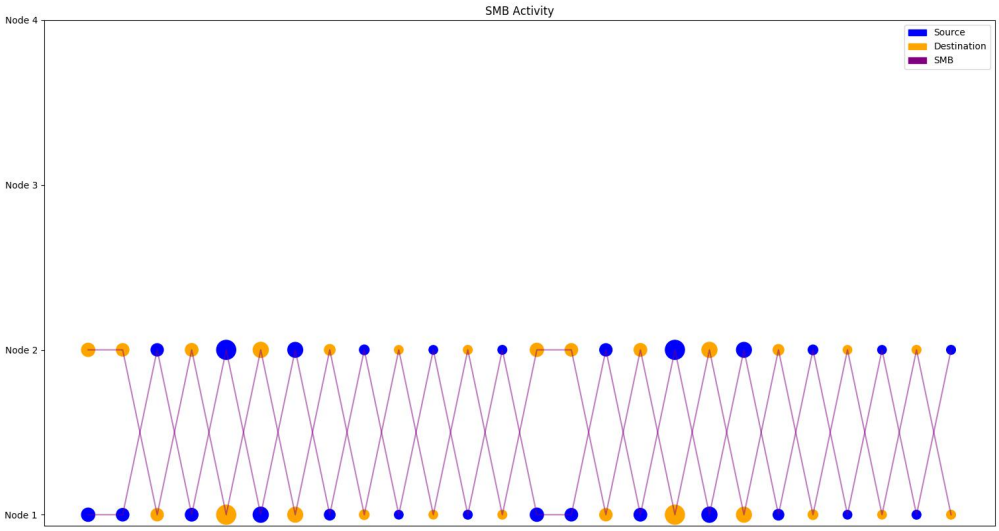


Figure 4 - WannaCry Wormless SMB activity

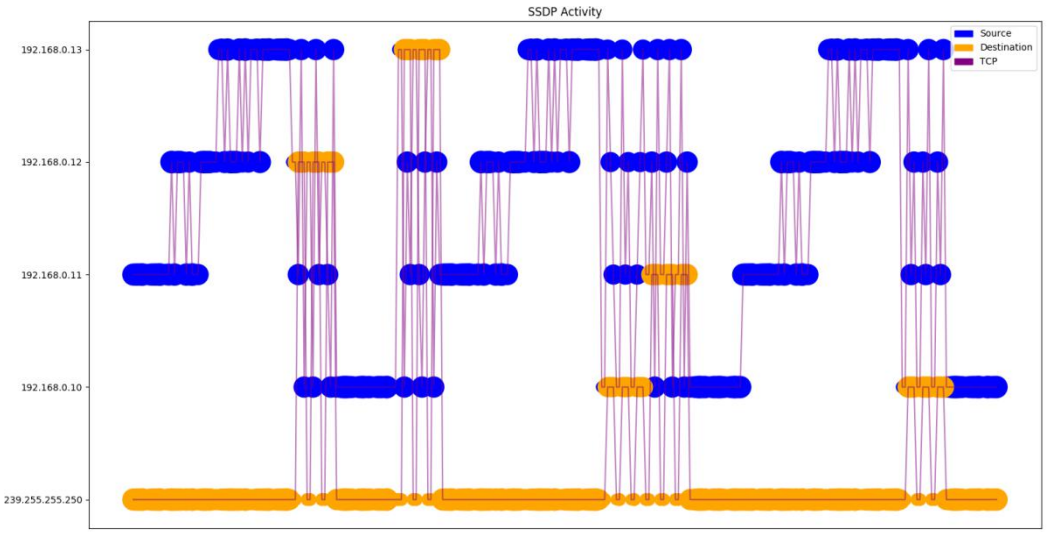


Figure 5 - WannaCry Wormless SSDP Activity

The SSDP activity (Figure 5) seems fairly typical, there is no obvious deviations in its behaviour. This would make sense, as one would believe that whilst the ransomware encrypts the files, the machine itself is still running Windows. Therefore even in its locked down state it would still perform standard network protocols. One could predict an obvious deviation in ransomware that works in a way that affects the overall operating system, like Petya / NotPetya.

Figure 6 shows the TCP activity across the network, an immediate observation is that for some reason Node 4 received and created no TCP activity. The reason for this is unknown, and could more than likely simply be a coincidence. What one can observe is a notable amount of TCP connections between 192.168.0.10 and 192.168.0.11, which are also known as Node 1 and Node 2. One will make note that the first round of TCP connections between 192.168.0.10 and 192.168.0.11 looks similar to the very first round of TCP connections captured between 192.168.0.12 and 192.168.0.11. This leads us to believe that this first round of communication is standard, but the third bulk TCP connections does not look similar to its predecessors, we believe that this is the connection captured during the SMB activity, since SMB works atop TCP. This is further compounded by the irregular constant TCP activity succeeding it, mirroring the SMB graph.

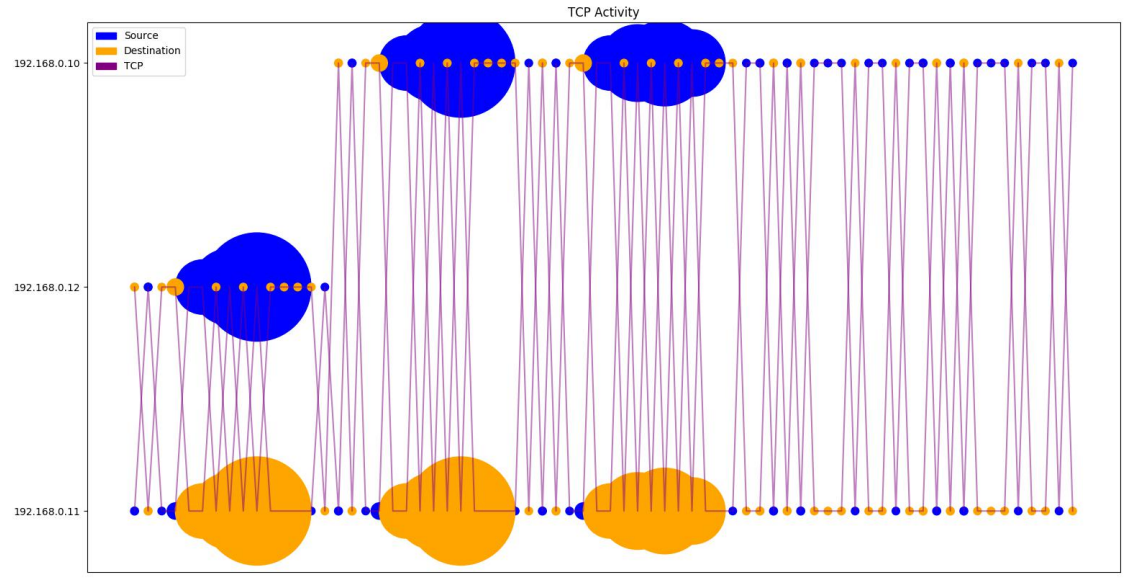


Figure 6 - WannaCry Wormless TCP Activity

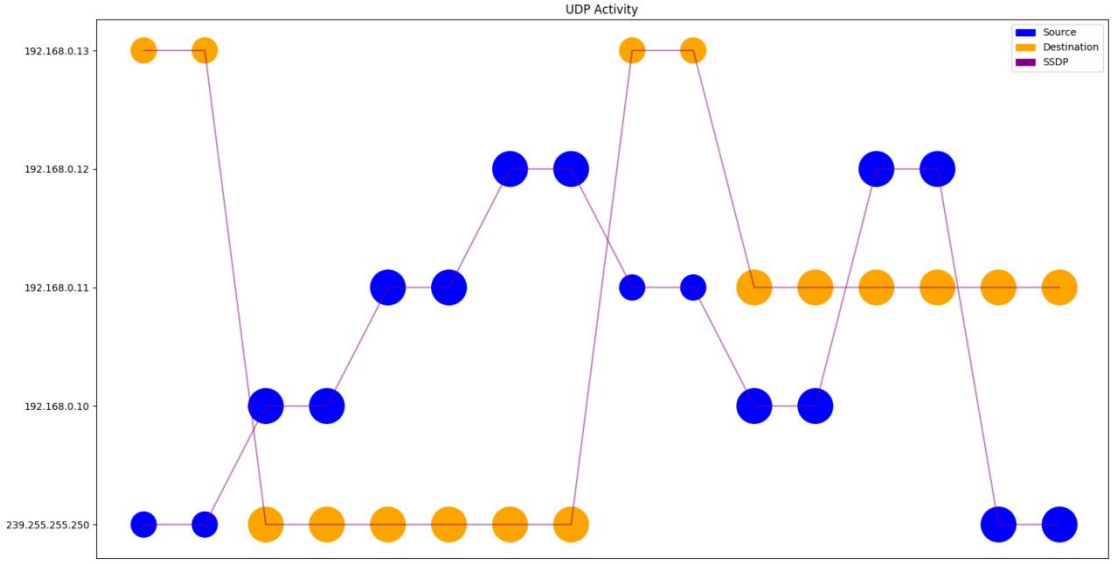


Figure 7 - WannaCry Wormless UDP Activity

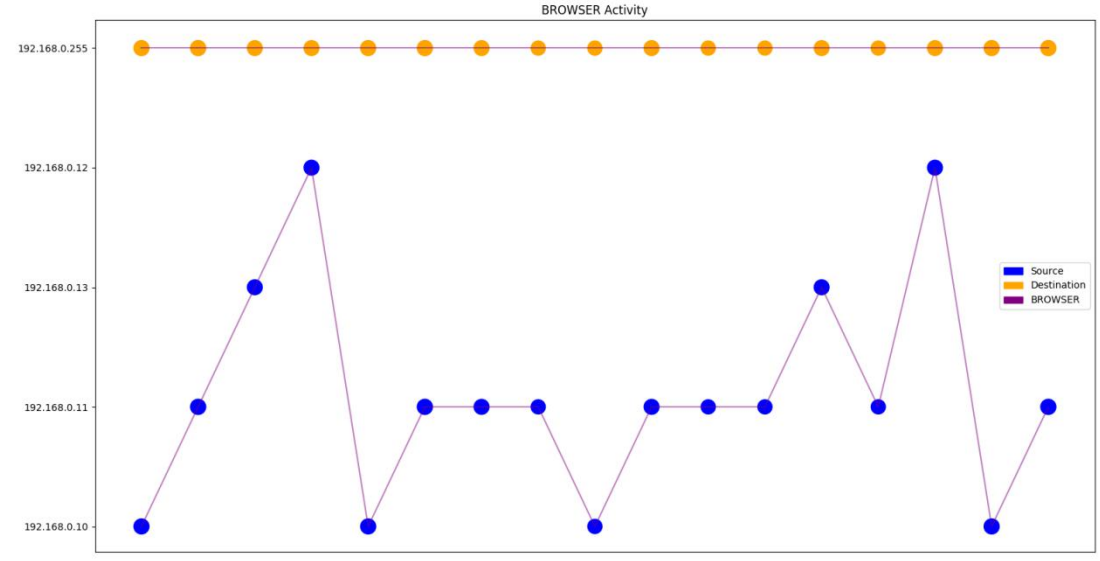


Figure 8 - WannaCry Wormless Browser Activity

The UDP and browser activity shown in figures 7 and 8 respectively appear completely typical, furthering the hypothesis that standard network activity tends to continue outside of the expected infected activity.

### 5.3 WannaCry

WannaCry (or WannaCryptor) was the first propagating sample we set out to observe and visualize. WannaCry caused much damage in 2017 before its killswitch was discovered by Marcus Hutchins, with the National Health Service UK (2018) reporting that “80 out of 236 of its trusts were compromised”, “603 primary care and other NHS organisations were infected”, and “1220 pieces of diagnostic devices were disconnected to prevent infection”. Its characteristics made it the perfect choice for being the first test, as it is renown for its ability to quickly and easily propagate using the SMB vulnerability EternalBlue and being generally very loud whilst doing so. The reason behind this is that after it has finished infecting the machine it was launched upon, it then begins probing the network for open SMB ports (445) across the network and if it finds one it will begin taking advantage of the EternalBlue exploit to propagate onto the machine. It will do this for every machine it can reach on a network, and there is even possibilities where the infected machine will propagate again onto already infected machines.

The first metric we decided to analyse was the CPU usage across all nodes in the network. The hypothesis being that since WannaCry is an encrypting ransomware, one should see an obvious increase in CPU usage on a single node, followed by subsequent spikes in CPU usage across various nodes as it propagates. Furthermore, due to the “ransomnote” running and WannaCry constantly scanning the network after infection, one would expect to see a general increase to CPU usage across all machines after infection. As one can see in figure 3, these hypotheses are mostly correct. One can observe the first 10 minutes of dormancy in the test with some erroneous activity from Node 1, possibly being caused startup programs attempting to update, or the node querying its own connection status.Then half way you can observe the very noticable spike in activity by node 1 (blue), this would be the launching of the WannaCry sample. After Node 1’s (blue) activity subsides, you can then observe a spike in activity on Node 3 (green), followed immediately by smaller spikes in activity on Node 2 (red) and Node 4 (orange). These first four spikes are the immediate infection and encryption points, all occuring in about the space of two minutes. Following these, you see a general increase in CPU usage across the board with numerous spikes in activity, which may be caused by WannaCry’s re-infection phenomenon mentioned earlier. Towards the end, activity returns to a stable, yet increased state of usage.

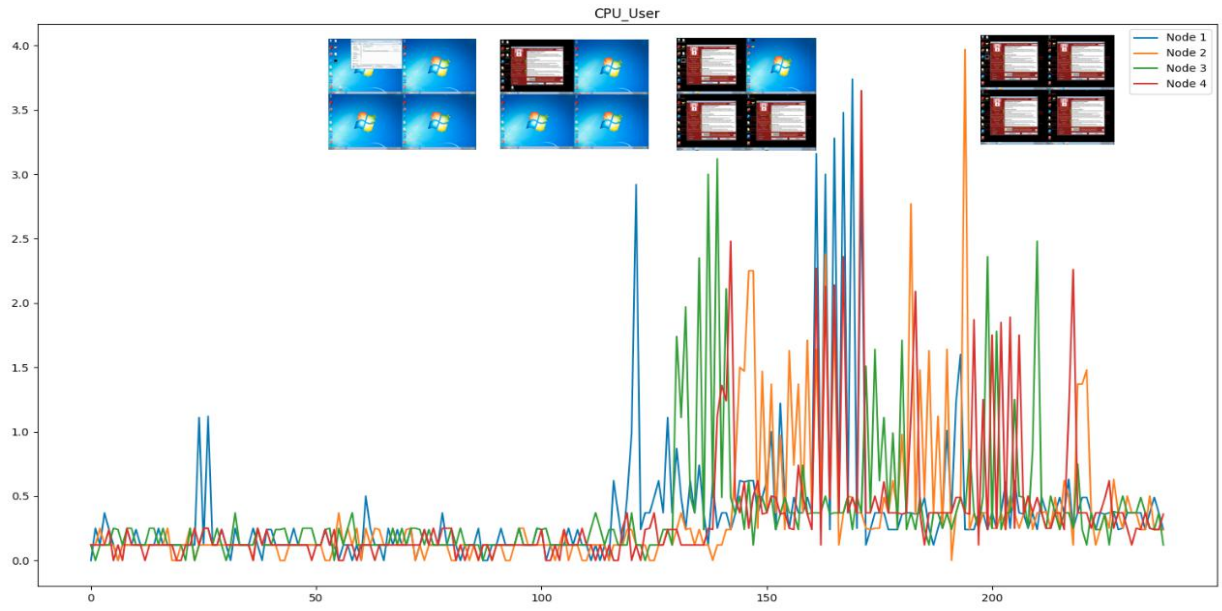


Figure 9 - WannaCry Network CPU Usage

Following this we can look at the network activity of the nodes, which is where things get fairly interesting. The first and most obvious thing to observe when dealing with WannaCry is of course activity involving the SMB protocol, in theory one should expect to see a fairly staggered increase in SMB communication across the nodes.

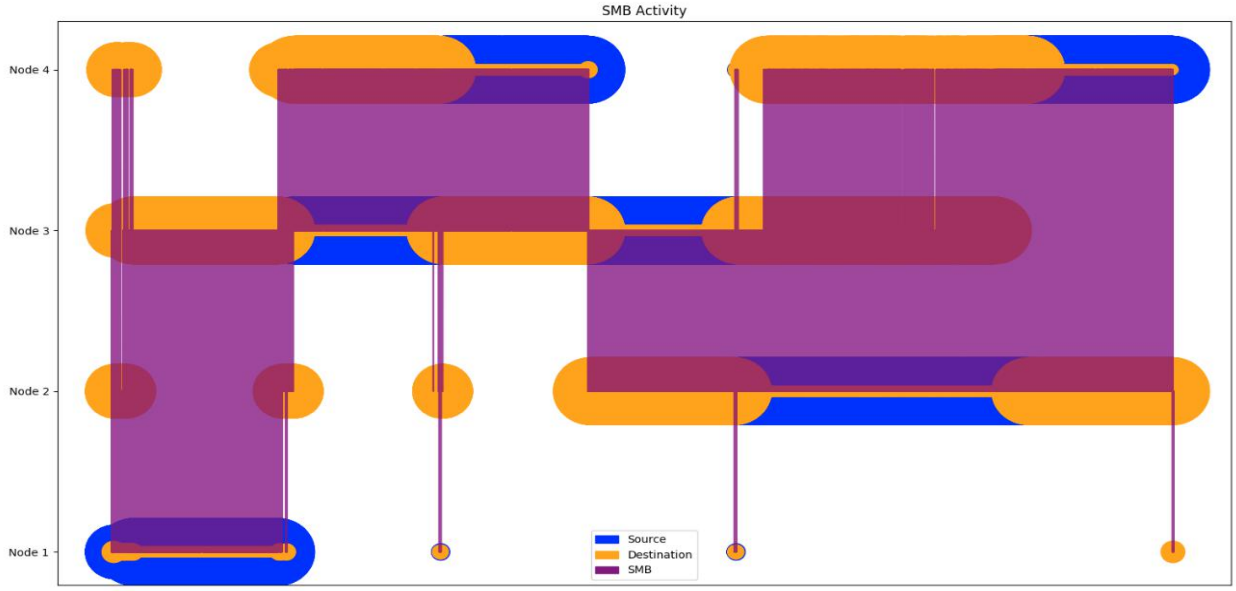


Figure 10 - WannaCry SMB Activity

The four rows each represent an individual node, with the orange colour representing when a node is the source of the SMB activity and the blue the destination of the same such activity, the purple lines represent where the SMB activity is going. The SMB activity of WannaCry is loud and constant, hence why this visualization in particular is also loud and perhaps confusing at first glance. What is happening is the initial infection occurs upon node 1 and propagates to Node 3, WannaCry then propagates from Node 3 to Node 4 and then to Node 2 (second row). All of the later SMB activity can be attributed to the infected machines attempting to reinfect the already infected machines on the network. One interesting thing to take away from this graph is the strange activity of Node 1, which communicates back and forth visibly with Node 3 for a time, and then ceases most activity after aside from the three points where it has a back and forth with Node 3 and Node 2. Why Node 1 doesn't display the continuous SMB activity akin to the other nodes is hard to say, could the initial infection of WannaCry only propagate in such a way once, and then leave the subsequent propagations to infect the rest of the network in order to project its entry point? One would believe this to be true if it were not for the fact that WannaCry even outside of its propagation is loud due to its nature as a ransomware.

The next protocol worth looking at is TCP, figure 5. At a glance one might notice that the latter half of the graph looks similar to that of the SMB graph distribution.

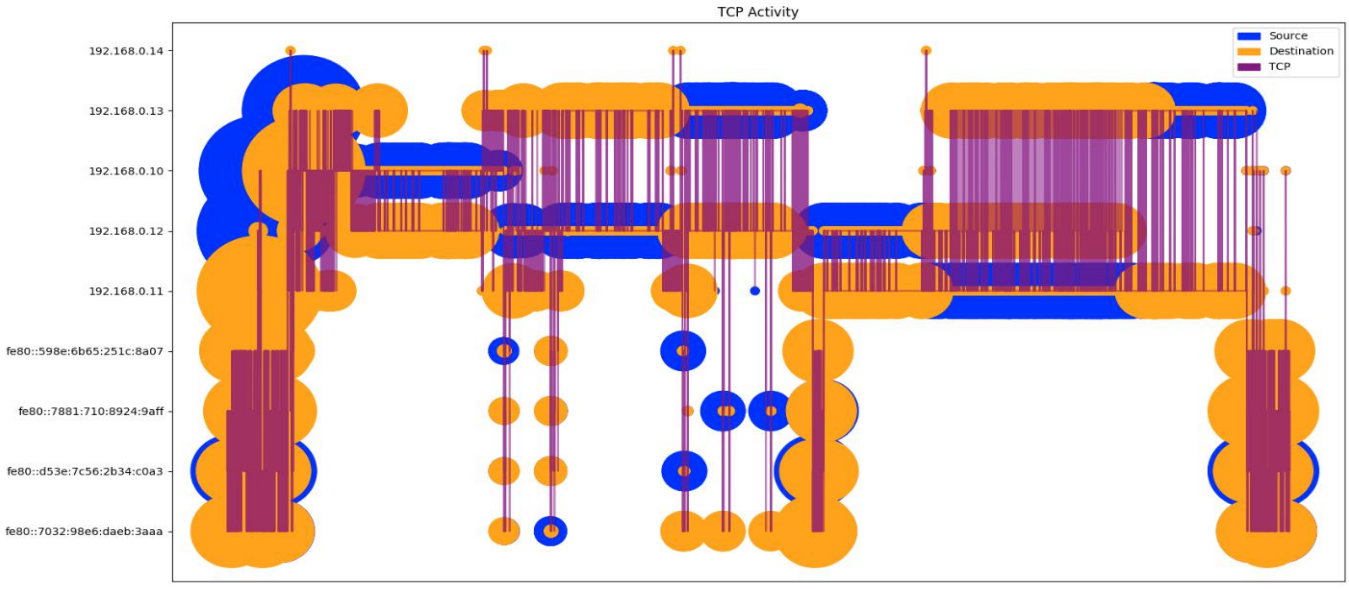


Figure 11 - WannaCry TCP Activity

This is to be expected, since the SMB protocol runs atop the TCP protocol on port 445, which this version of WannaCry is exploiting. Therefore the former half of the graph will be showing simply just TCP activity in general, and the latter half will be showing the general activity and the activity with the SMB protocol. However, the question therein lies if one can distinguish the difference between the two here, the answer being a rather resounding negative. The only obvious way of noticing the WannaCry activity is to have prior knowledge of the SMB activity, therefore this visualization on its own is not a reliable method of determining if propagation has occurred or not.

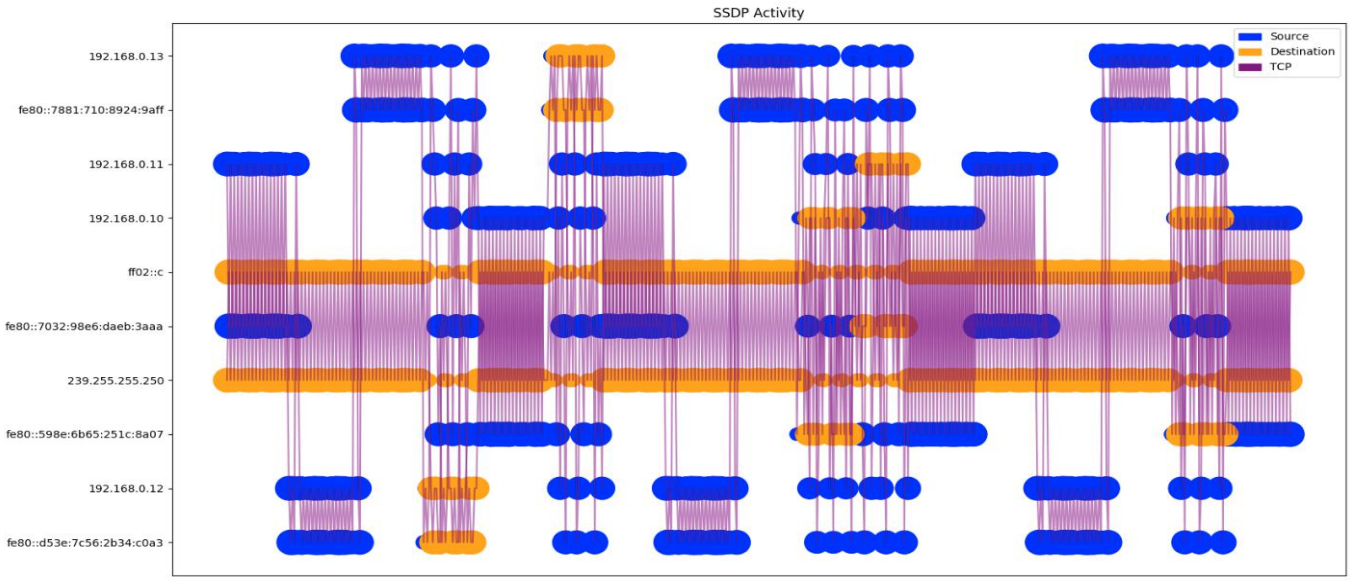


Figure 12 - WannaCry SSDP Activity

Nothing of particular note appears in the visualization of SSDP in a network infected with WannaCry, other than that it appears to consistently work as normal with very little deviations. One cannot pinpoint the exact point where propagation may be occurring or not. This is to be expected, since WannaCry is not known to utilize SSDP to any extent.

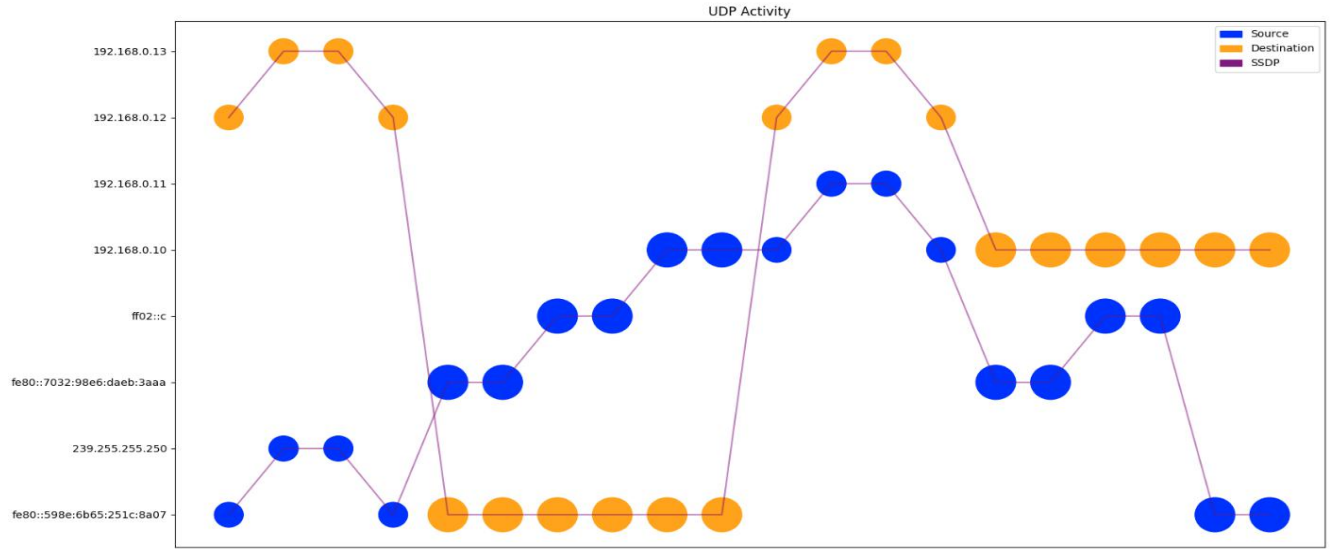


Figure 13 - WannaCry UDP Activity

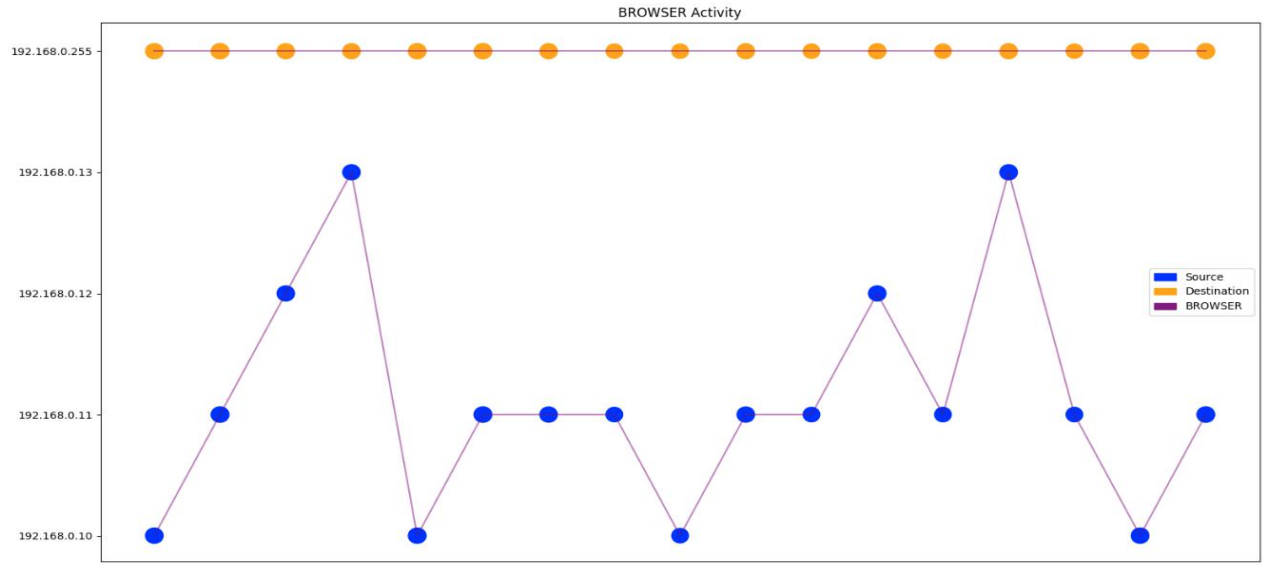


Figure 14 - WannaCry BROWSER Activity

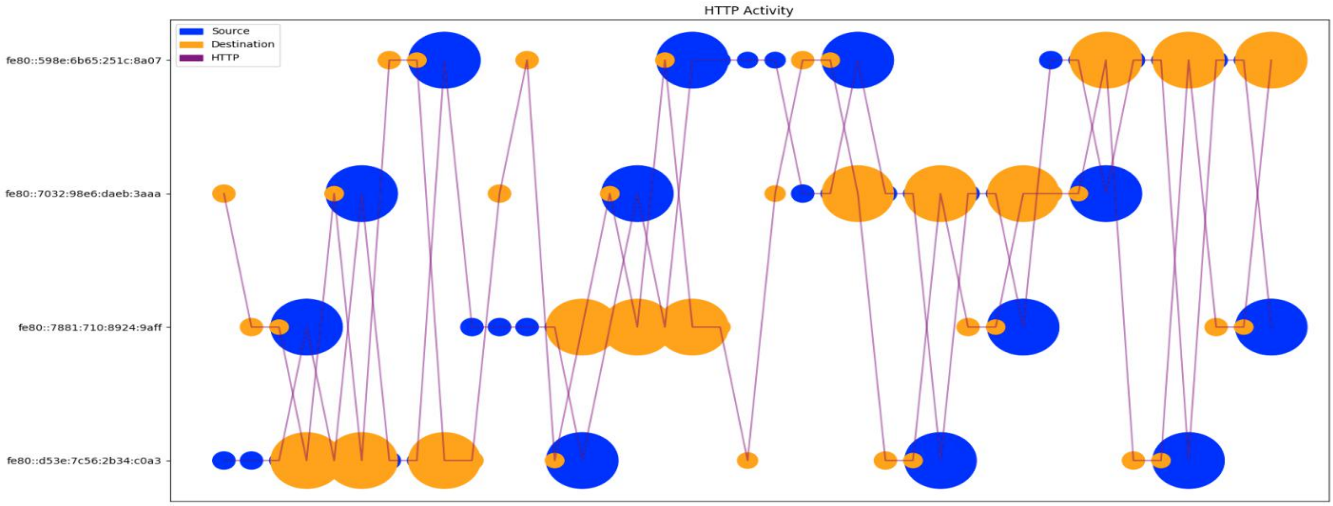


Figure 15 - WannaCry HTTP Activity

Just as with SSDP, there is no particularly obvious activity within UDP, BROWSER, or HTTP that allows us to observe propagation concerning WannaCry. All of these protocols appear to continue acting consistently even after infection.

Then finally we come to the screenshot gathering of our setup. The interesting thing here is that we can roughly estimate the time it takes to go from a seemingly clean network setup to being fully infected, happening in roughly two minutes and thirty seconds.

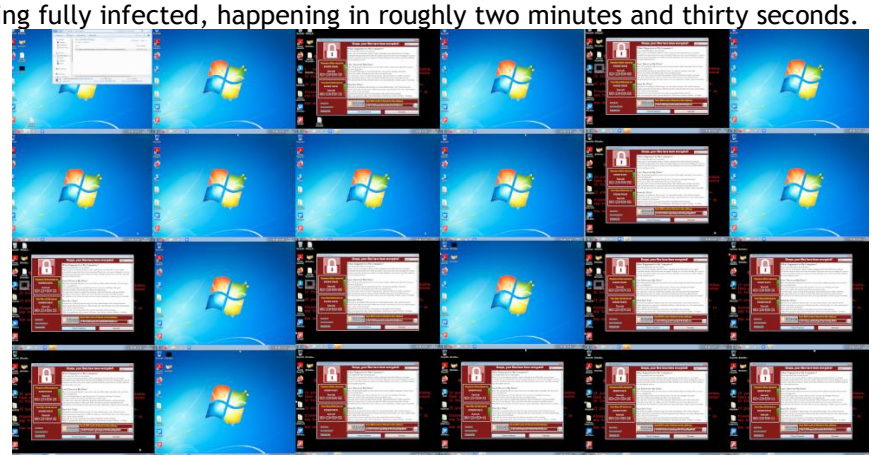

Figure 16 - WannaCry Network Clean to Infected

The final interesting thing that the screenshot functionality reveals to us about the nature of WannaCry propagation is something that has been mentioned at various points in this section, that being the tendency for WannaCry to reinfect machines it has already infected. This is proved when you look at the last screenshot taken in this test.

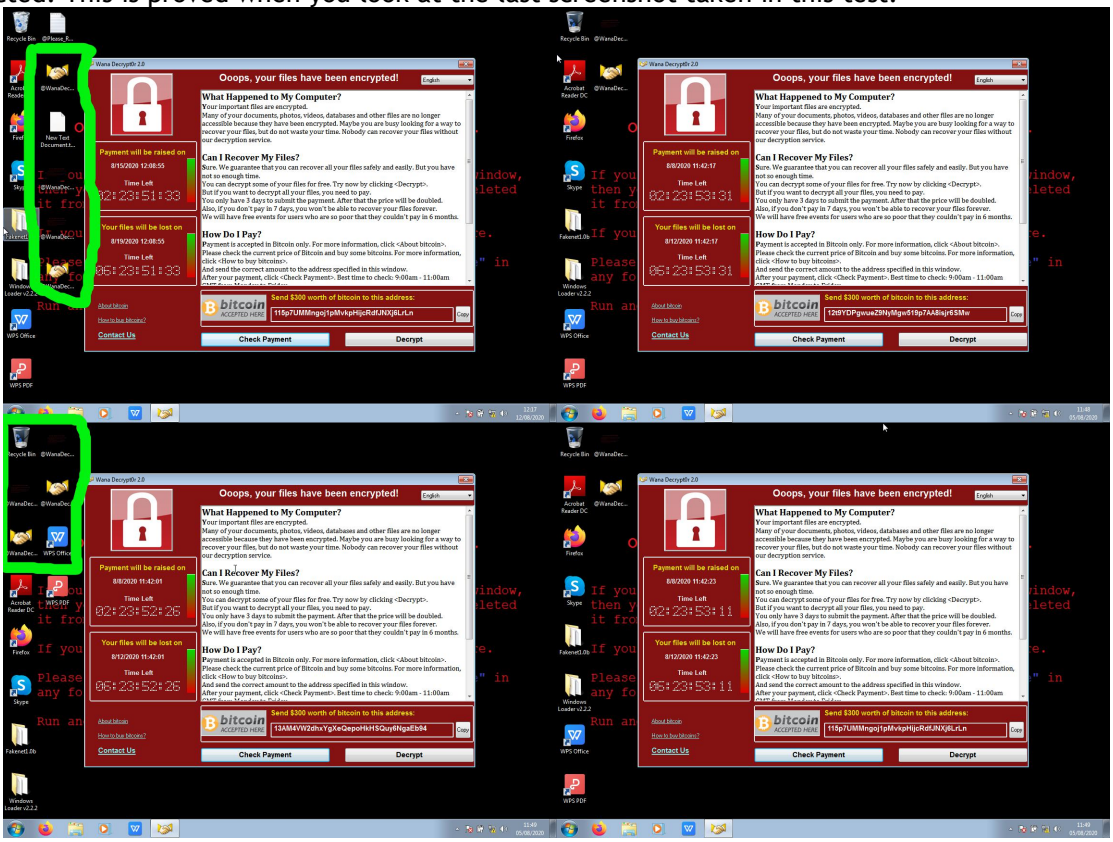


Figure 17 - WannaCry possible reinfection

Both of the nodes that were infected first have a second copy of the WannaCry executable on its desktop, aswell as an extra background which accompanies the executable. Full screen shots of each of these machines can be viewed in Appendix 1.

For further analysis of characteristics, a Cuckoo analysis was also performed on our WannaCry sample in order to reveal characteristics that our setup was incapable of detecting. Considering the sample ran and infected all four machines, what was not expected of Cuckoo to return was the possibility that WannaCry was actually infact contextually aware, this is evidenced by the fact that at some point in execution Cuckoo observed WannaCry specifically looking for the Cuckoo agent.py in the startup directory.

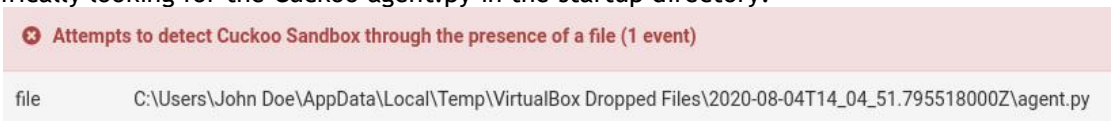


Figure 18 - WannaCry looking for Cuckoo

Despite this, the sample still infects and propagates across our LAN setup, and even more surprisingly, it still infects and encrypts the Cuckoo setup. This tells us that even though the WannaCry sample was developed to detect being in a possible sandbox environment, it doesn't act on it once detected. Another possible conclusion is that it does infact act on it, but in such a way that it encrypts the agent.py first and then moves on to the rest of the files in order to limit the amount of analysis and possibly stop the sandbox environment from detecting the EternalBlue SMB exploit.

## 5.4 CryptoLocker

Within this study we tested three variants of CryptoLocker, appearing in September 2013, November 2013, and January 2014. Each of the tests ended inconclusively with the payload of the CryptoLocker ransomwares not deploying within the twenty minute time frames. Following further research a unique reason for such behaviour became apparent, after being launched the CryptoLocker ransomware creates a process visible in task manager;

inserts itself into the startup programs; and then deletes the original executable that it used to create said processes. This was all observed, and the process of the ransomwares can be observed in task manager *(Figures 13 & 14)*.

However this was as far as the ransomware proceeded, no propagation was expected because CryptoLocker is not known to propagate internally in local area networks, but what was not expected was the lack of encryption, which is where our unique situation unfolds. After deleting the injection method, CryptoLocker then has to communicate with its Command and Control server, which is now known as the GameOver ZeuS server. When it communicates with the server, as well as registering it as part of its botnet, the server also generates the 2048bit RSA keys which it sends back to the clientside CryptoLocker in order to allow it to encrypt all of the files.

This presents two such interesting situations, the first and most obvious being that the GameOver ZeuS server was taken down in Operation Tovar in 2014, meaning that any communication with the server is now impossible. But if hypothetically the server was still available, the ransomware still would not be able to run because our testing setup works on an internal network structure, meaning it has no ability to communicate with the server even if it was available. Overall, this presents a previously unforeseen difficulty, that malware which requires external influence in order to perform its entire function will not be able to function fully in our setup. In essence, a flaw of the system created may be that it is only useful for testing dropper type malwares and not downloader types or botnets.

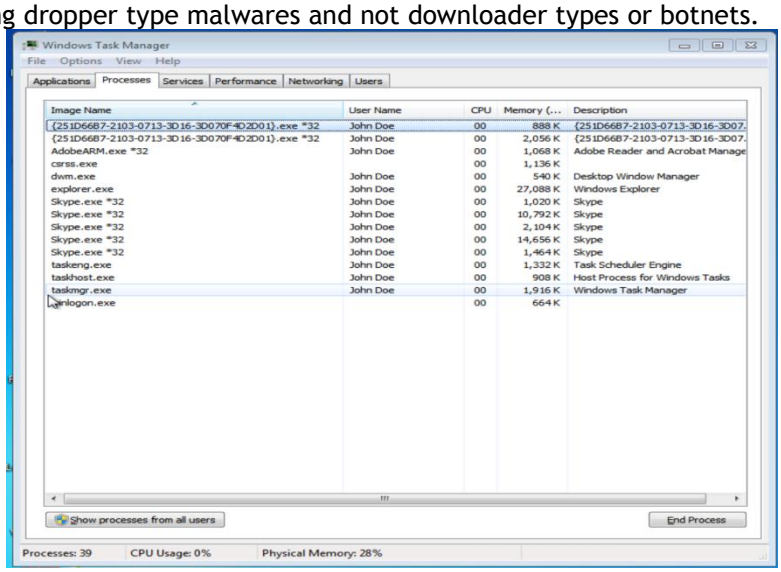


Figure 19 - September Variant Process

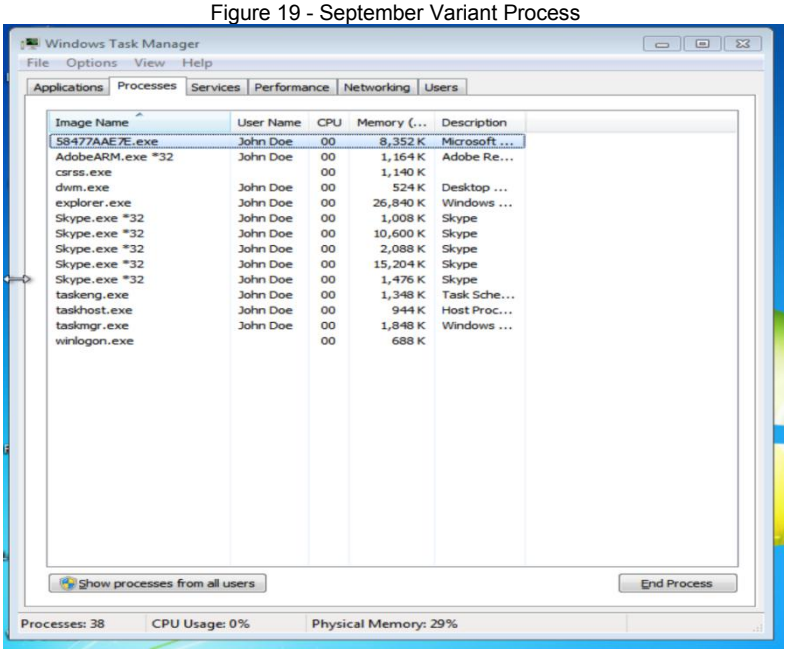


Figure 20 - CryptoLocker January 2014 Variant Process

## 5.5 Locky

Unlike CryptoLocker which presented an unforeseen difficulty in the testing of malware, Locky presented a predicted one. In essence, Locky behaves similarly to CryptoLocker in that after it is executed it performs self deletion. Where it differs is that Locky is known to be context aware, meaning it has multiple methods of identifying whether or not it is in fact running in a virtual environment or not. In our test, the sample performed observable self deletion and then nothing else happened, all of the reported infection identifiers were explored in registry and system files with none to be found. Therefore it is believed that this is the first case during testing where a sample was able to identify it was within a virtual machine and decided not to execute. There was a previous case where the WannaCry has a form of contextual awareness, according to a Cuckoo analysis in which it was spotted looking specifically for the Cuckoo agent in the startup directory, however it is assumed that this was a basic implementation of contextual awareness in order to avoid known sandboxes and is therefore able to execute in our setup because it is not looking for the telltale signs of a virtual machine, but only the signs of the Cuckoo setup (Figure 12).

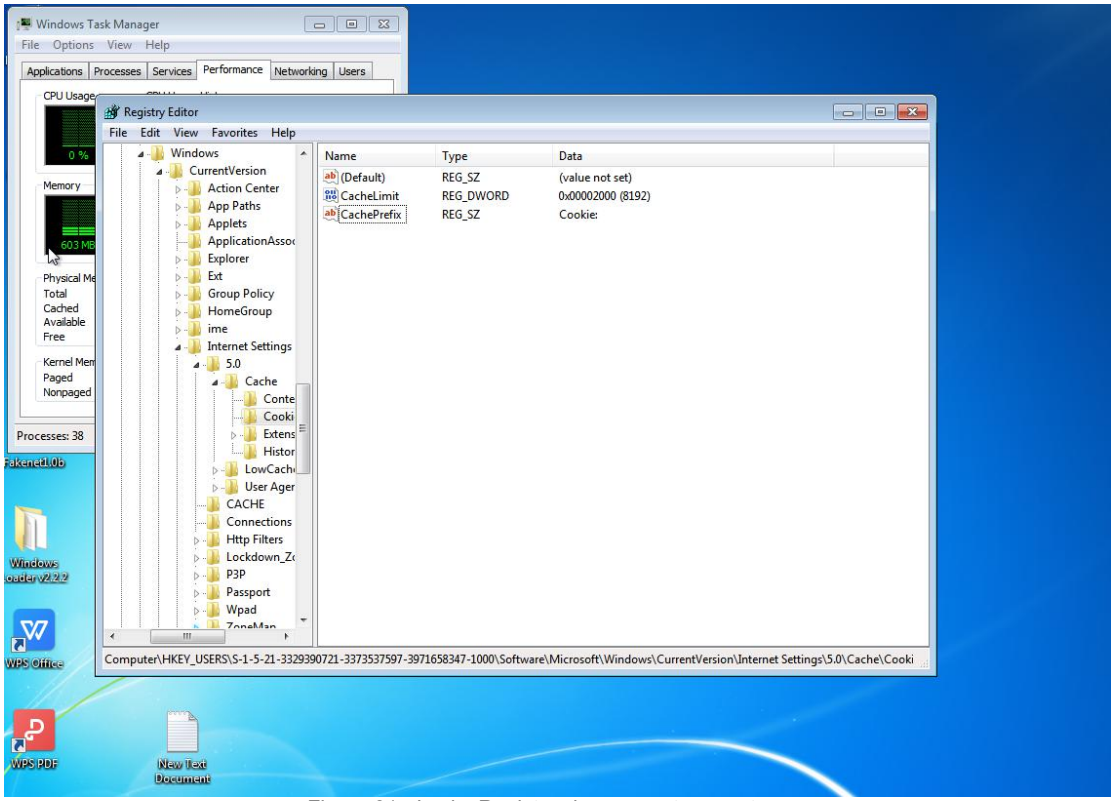

Figure 21 - Locky Registry changes not present

## 5.6 Petya

If one wants to observe potential propagation, then the variant of Petya to look at would be NotPetya / PetrWrap, since it wasnt until this later variant that the SMB Eternalblue vulnerability was exploited in order to allow it to propagate. However as with WannaCry, we though it necessary to visualize the known non-propagating version in order to compare it against the version that is supposed to. Petya is a ransomware malware, but it works a quite a bit differently to the standard ransomware one might be used to. With the standard ransomware, it will encrypt your files and leave the operating system itself relatively untouched. Petya will infect your master boot record in order to boot the system into the ransomware instead of the operating system, from here it will then encrypt all of your files and display a ransomnote. Knowing this, we could expect multiple sharp increases in CPU usage due to the system becoming infected, rebooting, and the encrypting the files.

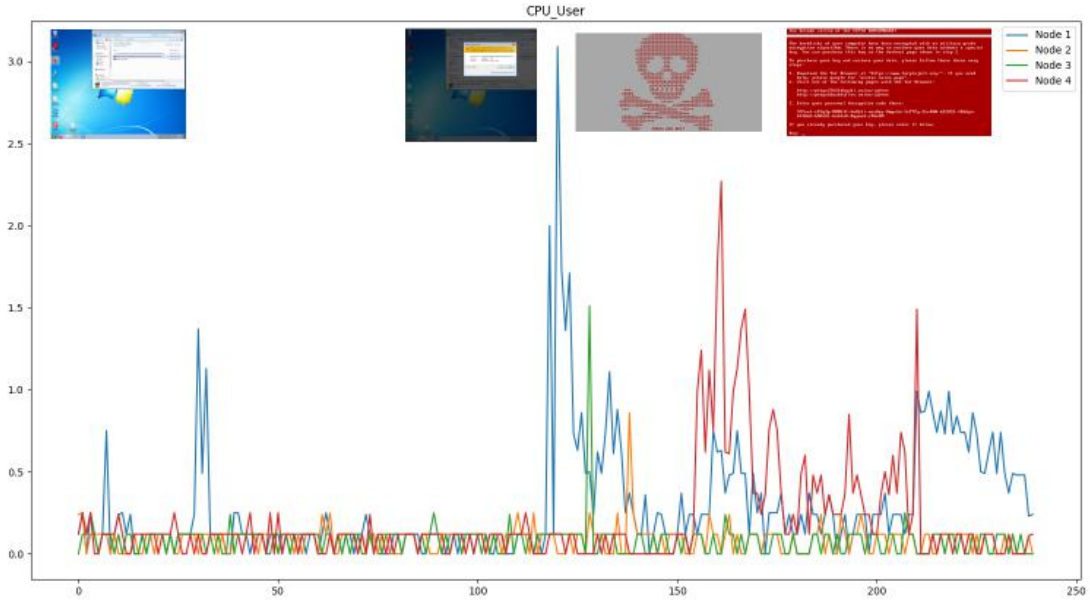


Figure 22 - Petya CPU Usage

The CPU usage is particularly interesting for two reasons, first of all it differs fundamentally in activity to that of WannaCry. Whereas WannaCry displayed an initial increase in activity and then a further increase in general activity thereafter, Petya shows an initial increase and then it very sharply trails off. The CPU activity is certainly still heightened, but it doesn't display WannaCrys tendency to consistently spike in activity. We believe this is because the standard ransomware might still be working in the background but Petya once it reaches the stage where it has restarted the system and encrypted the files and master boot record doesnt necessarily need to continue working, when it reaches this point the user is already entirely locked out of the system and the Petya ransomware cannot perform any functions that would allow to to propagate so the activity decreases. The heightened activity on Node 4 in this test is unexplained, the sample of Petya we used doesn't have the ability to propagate, we consider this activity to be erroneous.

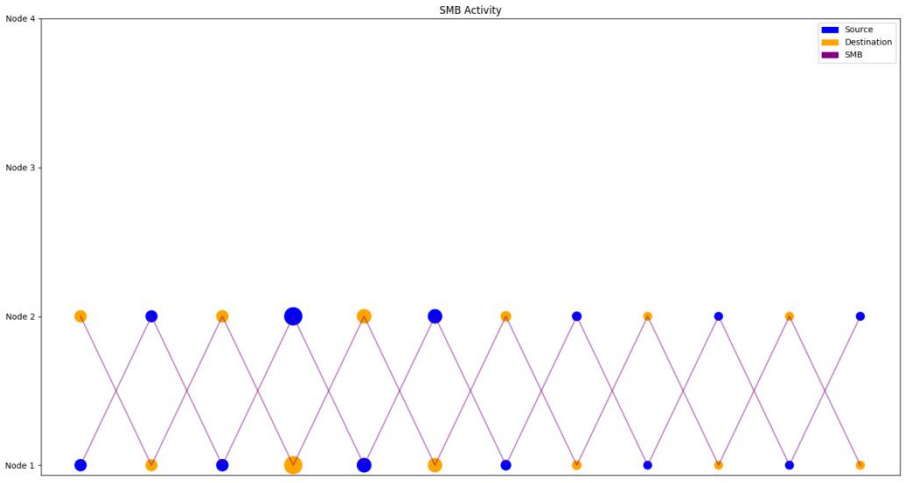


Figure 23 - Petya SMB Activity

The SMB activity originating from the environment is nearly exactly the same in this test as it was during the 5.2 WannaCry [Wormless] test. In that test, we theorized it possible that the whilst the WannaCry sample doesn't have the SMB EternalBlue worm in it, it could have the beginning of the implementation for it. Seeing the results originating from this test, we believe we can discount that theory, and instead accredit this strange activity to some erroneous SMB activity originating from our setup itself rather than the samples creating it. This does raise this issue of this erroneous SMB activity possibly interfering with our analysis of SMB activity from actual EternalBlue using samples, but we believe this wont be entirely

detrimental, since as one can see from the 5.3 WannaCry analysis, when a sample does utilize SMB to propagate, it is extremely loud and generates a lot of activity.

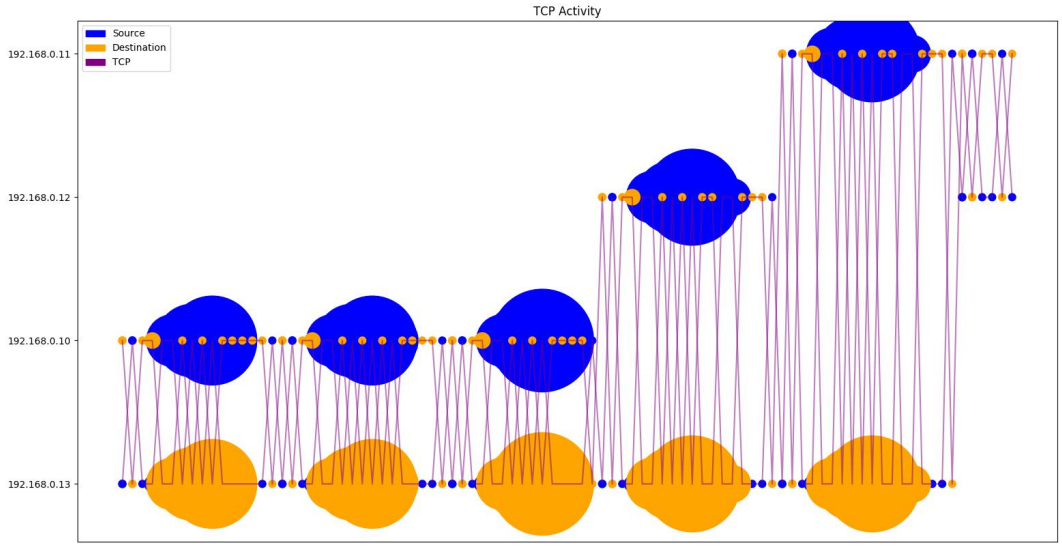


Figure 24 - Petya TCP Activity

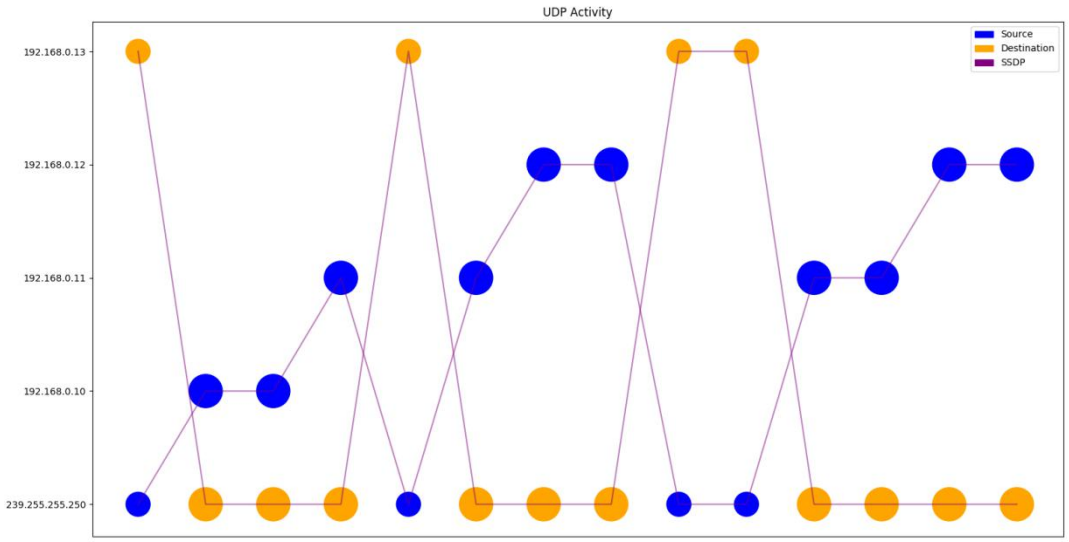


Figure 25 - Petya UDP Activity

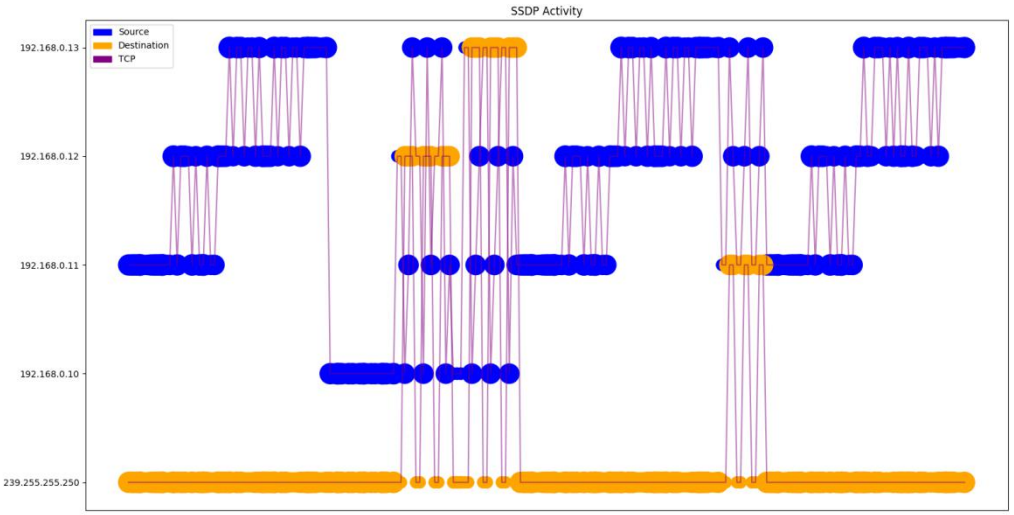


Figure 26 - Petya SSDP Activity

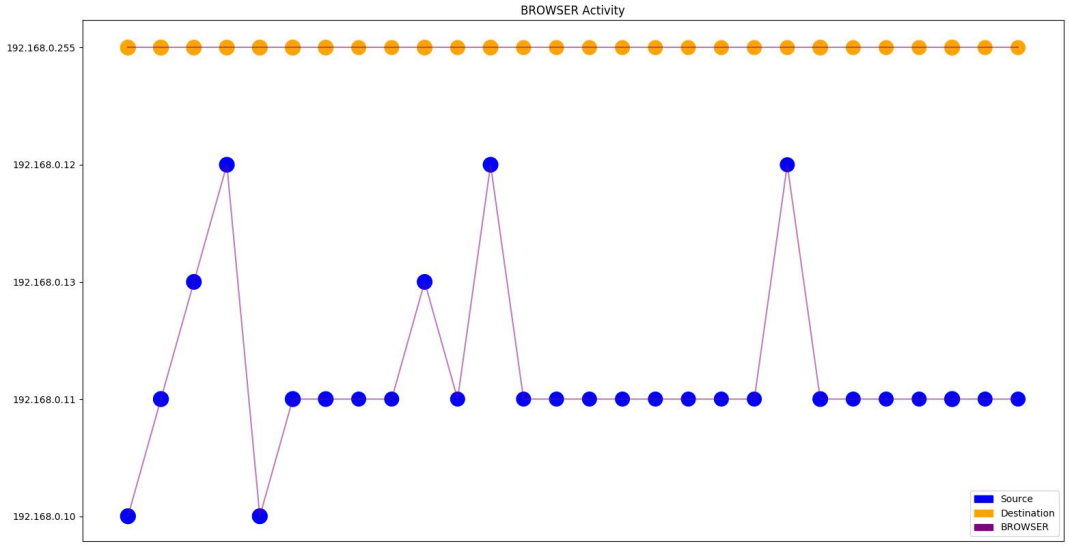


Figure 27 - Petya BROWSER Activity

As with the previous visualizations, there is very little of note to observe within the TCP, UDP, SSDP, or BROWSER activity that occurred during the test. If one were to conduct a deeper analysis of the network activity, one might be able to find activity that is a deviation from the norm, but at a glance, one cannot observe deviations that occur between the point in time where there is normal activity, and the period of time where the sample has been activated and the environment is experiencing an active threat.

## 5.7 NotPetya / PetrWrap

This is a variant or Petya that included various exploits that allowed it to propagate across networks, including the SMB vulnerability Eternalblue. But this isn't its only propagation vector, it also utilized a backdoor planted in M.E.Doc accounting software, and EternalRomance which is a ported form of EternalBlue. Using these propagation vectors NotPetya affected many different systems all throughout Russia and Ukraine. As a result, NotPetya is reknown for its aggressive propagation, which is interesting, because we tested two variants of NotPetya in our environment and whilst both infected Node 1 successfully neither achieved any form of propagation leading to the infection of the other nodes.

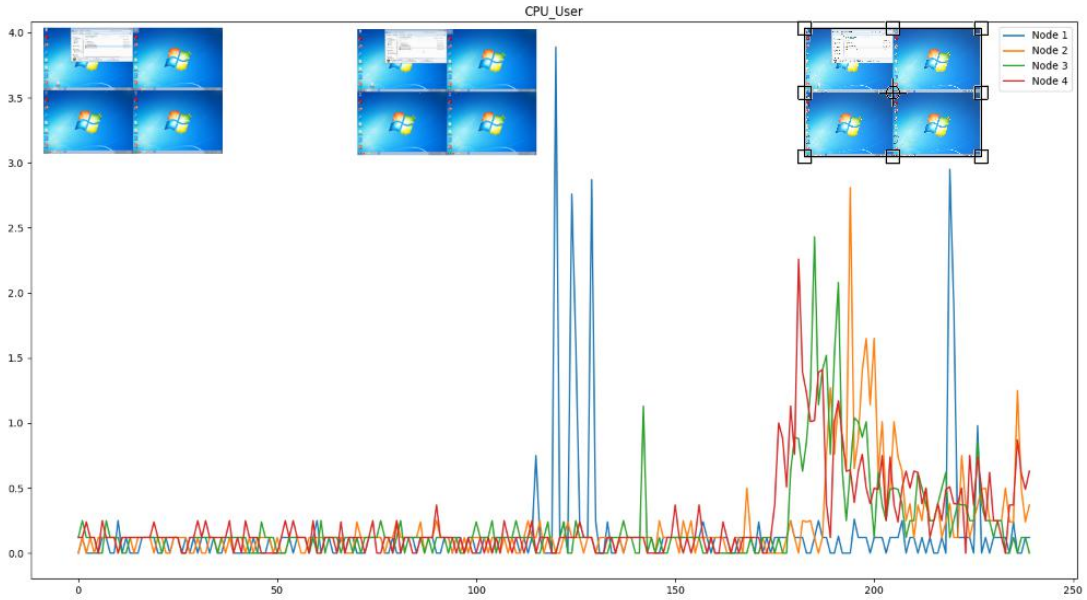


Figure 28 - NotPetya CPU Activity

The CPU usage captured during the testing of NotPetya was extremely interesting, because if one were just to compare the screens prior to and after the activation of the malware sample, one could not tell any difference. Yet, underneath there is quite the obvious point of activation and furthermore clear evidence of the malware continuing activity after its activation. Another interesting thing is the spikes in activity on the other nodes, the malware did not appear to propagate after activation, yet the cluster of activity presented to us in this visualization quite clearly indicates something happening. If this activity had occurred from a singular node, it would have been reasonable to dismiss it as an erroneous spike in usage, but a cluster of very similar spikes in usage appearing in all of the nodes except the initial infection node is suspicious indeed.

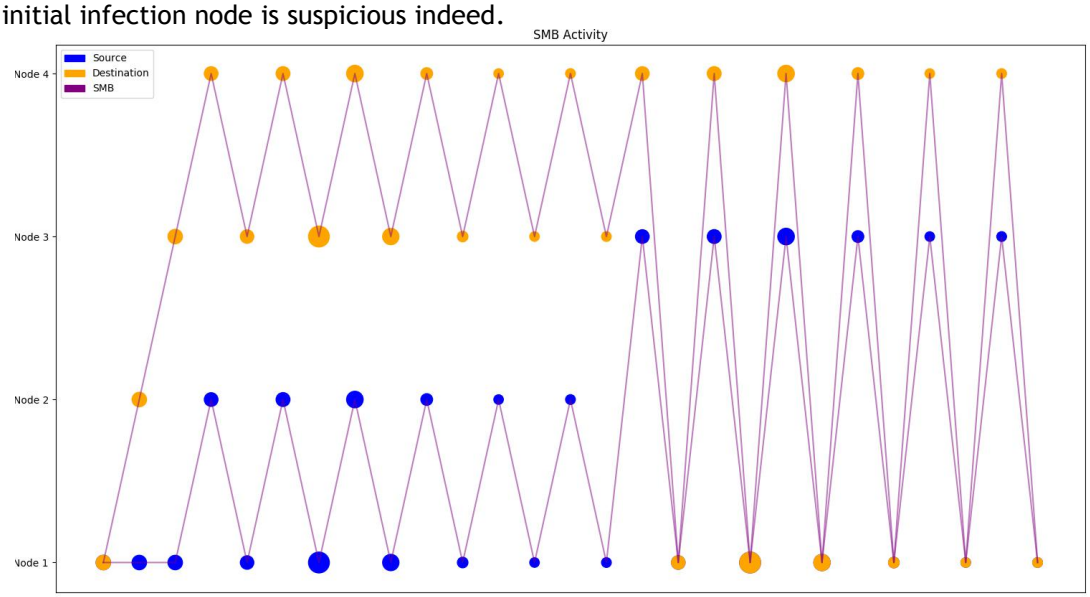


Figure 29 - NotPetya SMB Activity

All of the above SMB activity takes place post activation of the NotPetya sample, and therefore is a clear indicator of the initially infected node attempting to establish communications with the other node. This graph sits in place between the SMB graphs that show little to no activity and those that show much activity. Comparing it against figures in sections 5.2 and 5.6, one can clearly see an increase in activity across the board, but comparing it to the SMB figure in 5.3, one can see that the sheer amount of activity isn't even comparable. This could mean one of two things, either the sample was searching for open SMB ports in order to propagate later, or for some erroneous reason the sample didn't propagate.

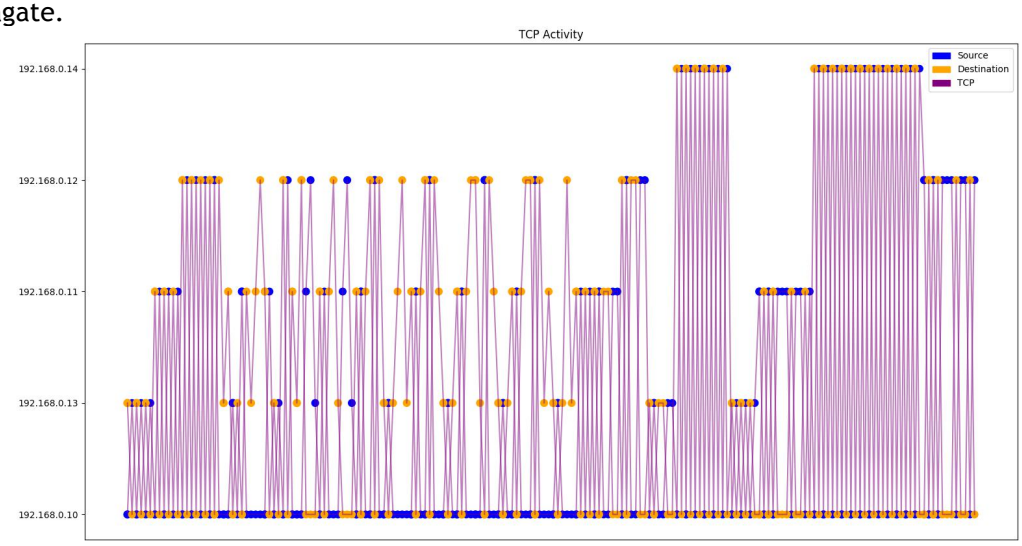


Figure 30 - NotPetya TCP Activity

It is once again worth noting that all the above activity takes place post infection. The increased levels of TCP usage is incredible and clearly shows us the Node 1 (192.168.0.10) is infected since it is generating an absolutely obscene amount of TCP packets whose destinations eventually account for every machine across the network, including our Ubuntu network monitoring node (192.168.0.14) which wasn't found during the testing of 5.3 WannaCry. This would reinforce the characteristic identified by many before us, that NotPeyta is extremely promiscuous and will attempt to spread to any machine regardless of what it is running, so long as it has a port it can possibly spread to.

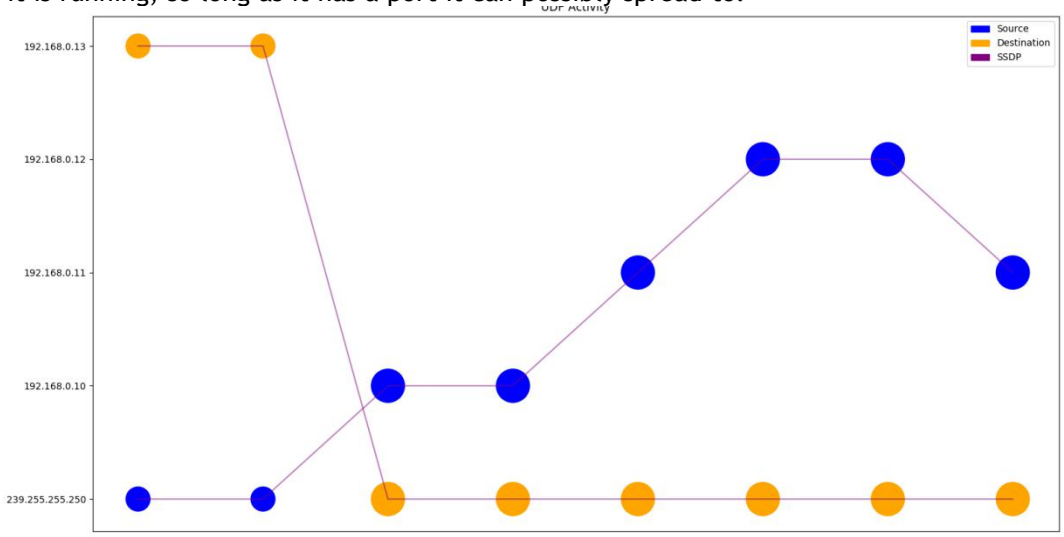


Figure 31 - NotPetya UDP Activity

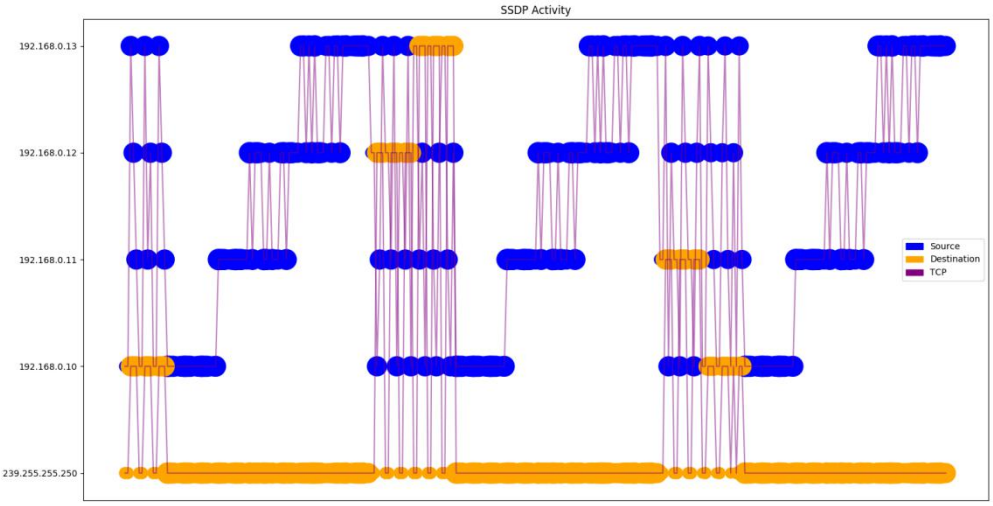


Figure 32 - NotPetya SSDP Activity

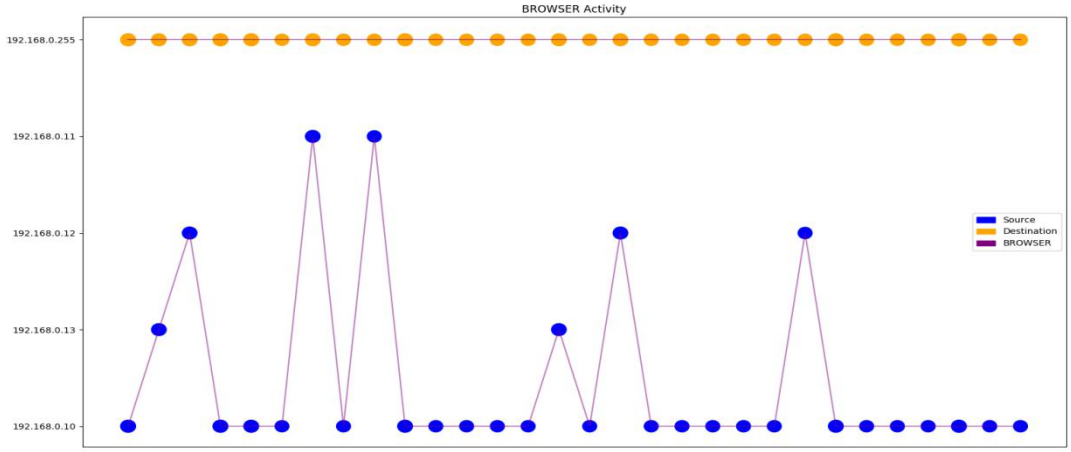


Figure 33 - NotPetya BROWSER Activity

Once again, as in previous iterations, there is no deviance in the standard activity of various network protocols.

The final piece of evidence to supply here is to prove that Node 1 was actually infected during the test. NotPetya has an interesting characteristic that makes it entirely different from Petya, in Petya, after the machine is infected it immediately reboots so it can infected the master boot record and encrypt your files. However once this is done, the machine cannot then create any network activity, and therefore in the case of NotPetya, cannot propagate. Therefore, NotPetya delays the restarting of the system by an hour by instead creating a scheduled task for the reboot, giving the malware time enough to propagate. This scheduled task was visible in the testing of both samples, figures 34 and 35.

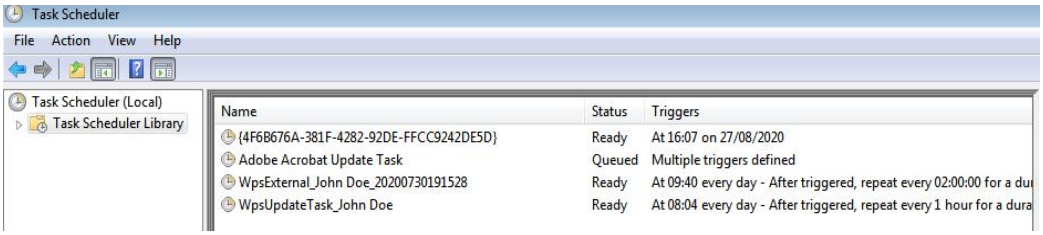


Figure 34 - NotPetya, Test 19, Scheduled Restart

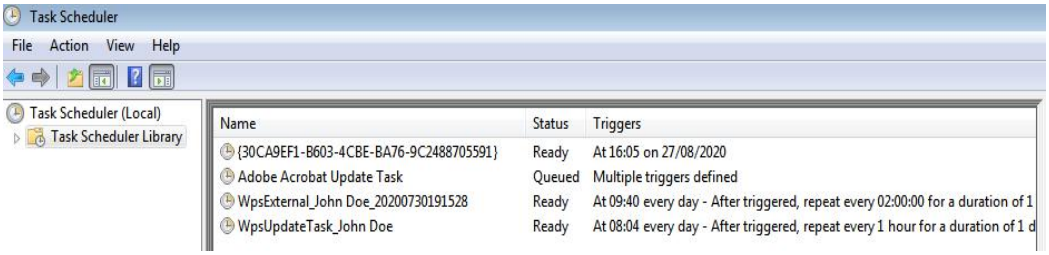


Figure 35 - NotPetya Test 20, Scheduled Restart

All of the visualizations and evidence supplied in the section poses one overall question, if the initial machine became infected and generated suspicious traffic that made it look like it was intending to propagate, why didn't it? There are a few possible conclusions here, the first and in our eyes, the most unlikely, is that all of the data that looks to imply an attempt or a start at propagation is a red herring, and the sample was simply searching for other machine with no intention to propagate. The second, relates to both the nature of NotPetya and our testing setup. NotPetya is known to work in a delayed fashion in order to allow itself to propagate, but also to not raise too much suspicion in the user of the infected machine, therefore it is possible that the initially infected machine scouts out potential candidates for infection, only to infect them at a time closer to which it will restart the system. This would mean that the actual propagation of the NotPetya malware would occur outside of our twenty minute testing window, and therefore is unrecorded.

We believe the latter conclusion to be entirely possible, and also testable, therefore the following test will take a different form, we will test the sample until the initially infected node reaches the point where it is rebooted and therefore incapable of performing propagation and analyzing the results.

## 5.8 NotPetya Extended

As aforementioned, NotPetya has been observed to act in a peculiar way compared to other ransomware, which is entirely derived from how it operates. Since Petya holds ones machine captive by forcing it to restart and then encrypting your master boot record and files, it makes propagation difficult, since propagation via EternalBlue is much more difficult if not impossible when not running inside the operating system. Therefore, the NotPetya variant bides its time, after the initial infection it finds machines in its network but doesnt immediately act, then after a certain amount of time it infects the other machines. When it infects the other machines, it puts in place a similar forced restart time to the initially infected host in order to enact a coordinated lockdown.

Our tests had to be adapted for this particular variant of malware, and after allowing it to work for its entire attack timeline, it successfully propagated to all of the machines on our virtual network.

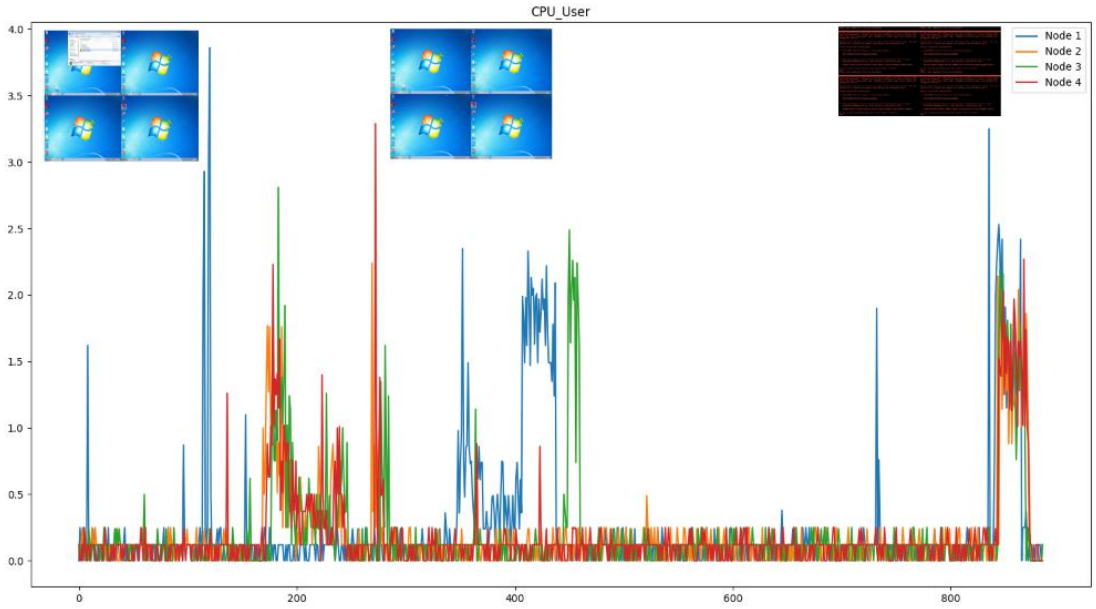


Figure 36 - NotPetya Extended CPU Activity

The CPU metric usage for NotPetya over roughly an hour and ten minutes is extremely interesting, you can clearly see where the infected activity takes place, which Node 1 being the launching point for initial infection and then the other machines gradually being infected after. But arguably the most interesting thing is that after all the activity that would indicate infection, activitiy appears to return to the normal levels they were at before Node 1 was even infected. It then isnt until the coordinated restart occurs that activity once again spikes, signalling what is obviously the encrypting of files. Its entirely possible that the spike in node 1 activity during the normal activity streak is the initially infected node checking for additional systems, or trying to infect our unreachable network monitor node.

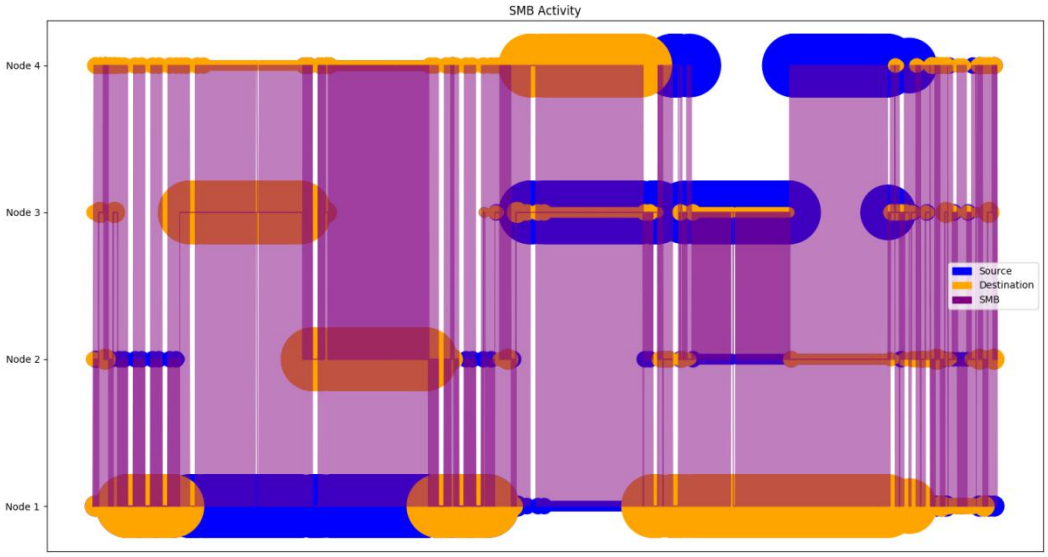


Figure 37 - NotPetya Extended SMB Activity

All of the SMB activity in figure 37 occurs after the ten minute infection mark. It is worth making the comparison of this figure to figure 10 which shows the SMB activity of WannaCry. The amount of SMB activity generated by the infected machines is very similar, the only way in which NotPetya appears to differ is that the activity is much denser and happens much more frequently, but such an observation could be obfuscated by the fact the duration of this test was a whole fifty minutes longer than the WannaCry tests.

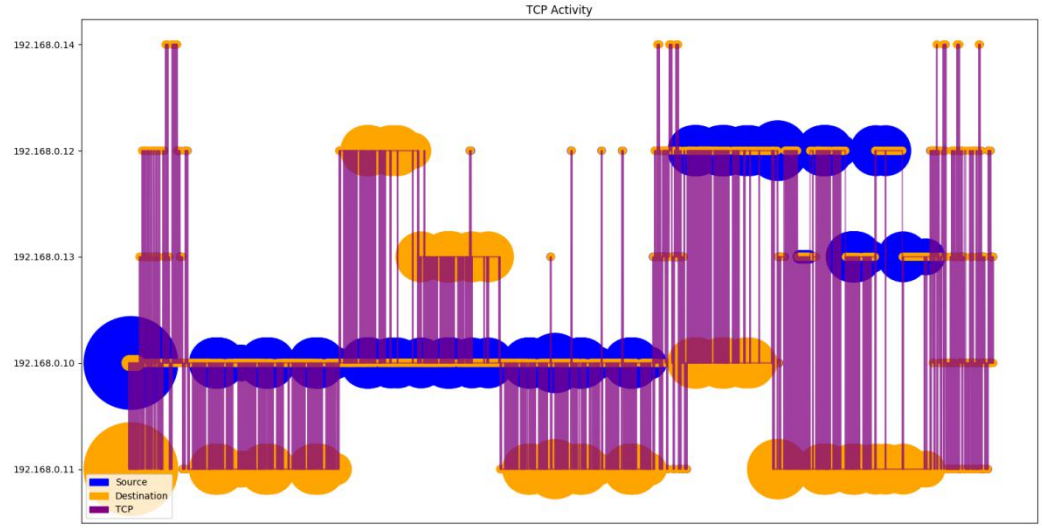


Figure 38 - NotPetya Extended TCP Activity

Interestingly, the TCP activity from the extended testing looks entirely different to the TCP activity that occurred in the regular testing, but once again shares more characteristics with the TCP activity that is displayed in Figure 11 which shows WannaCry TCP activity. As in figure 11, there are many clusters of consistent activity across the board.

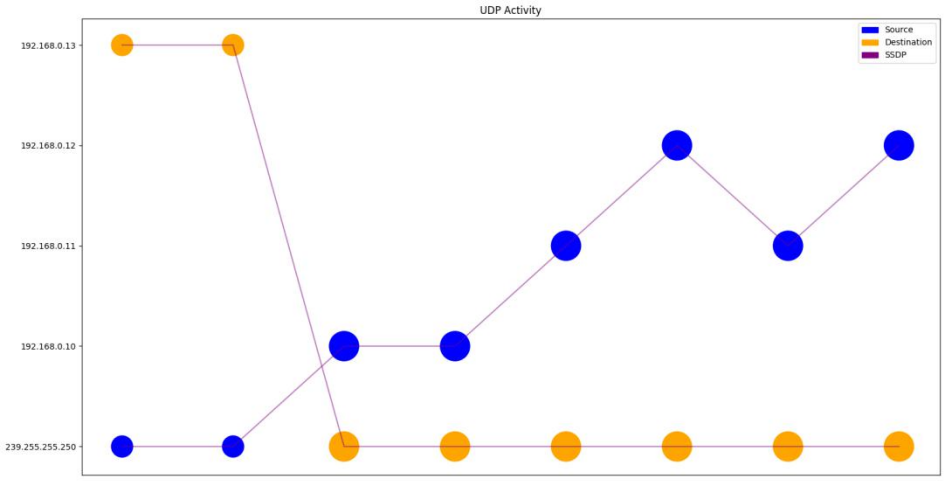


Figure 39 - NotPetya Extended UDP Activity

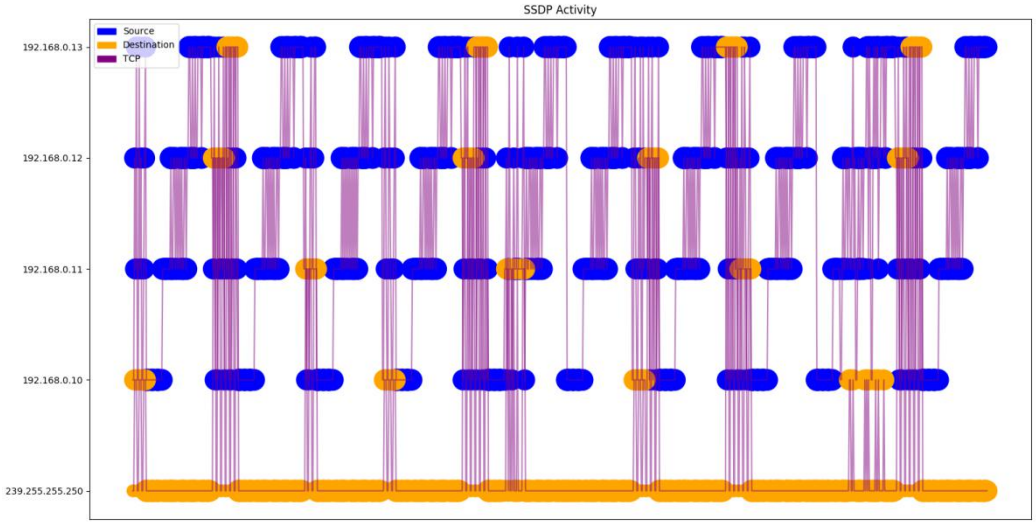


Figure 40 - NotPetya Extended SSDP Activity

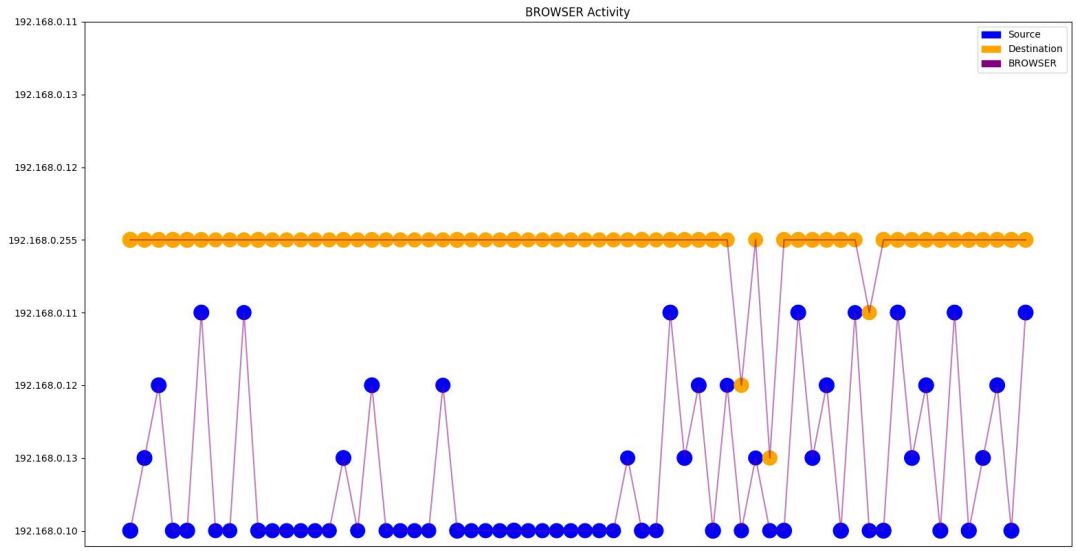

Figure 41 - NotPetya Extended BROWSER Activity

As with previous tests, the standard protocols show very little of note other than consistently normal activity pre and post infection. What these figures obviously will not show is the sudden cut off of activity after the machines have been forced to restart by the NotPetya infection, which would probably be pretty pronounced and obvious. The fact that this is missing is an acknowledged flaw within our visualizations and will be works on to be improve in later versions.

There are a few key points to take away from this extended test, the first and most obvious being that our established testing method is inadequate for testing all forms of malware. If we had concluded the NotPetya testing with the original tests, the conclusions would have been that our setup was for some reason inhibiting propagation, or the malware itself was not propagating, be it because its missing the component for propagation or because it is contextually aware. Our tests were designed as they were because of the desire to test quantitatively, it is becoming more and more prevalent that it might be a better idea to test malware qualitatively in order to get a better understanding of their operations, and tailor each individual test to encourage the malware to fully deploy its payload.

# 6 Discussion

In this section we will discuss the results in relation to the goals and objectives we established early in this paper in section 2.1. As well as discussing those, we will also explore any additional observations that were made and reiterate any of the particularly notable findings from the testing stage.

## 6.1 Visualizations

Our first goal was to create clear and interpretable visualisations of propagations. During development this was adapted into a broader sense but still achieves the same goal. Instead of specifically looking to visualise the observed propagations, we instead looked to gather metrics and logs to visualise and draw conclusions from of whether propagations occurred instead of outright simply visualising the way in which a malware observably propagates. This shift in focus allowed us to play more into our other objectives, in our original method effectively communicating the propagations effect on the overall network wouldn't have been possible since all we would have been visualising is the route which the malware took, rather than the metrics and network activity that lead up to the full propagation of the malware.

In regards to creating clear and interpretable visualisations we do believe we have been successful. Each of our visualisations each display a particular relevant metric in a format that best suits it. For example, it was decided the best choice for representing the CPU usage across the nodes as a multi-line chart, which made the most sense to us since it is a simple and effective way in which to communicate a change in value across time. Furthermore, it

was decided to present the values together rather than separately as having them together allows for the easy comparison of values. The idea of this was taken from Raffael Marty's (2008) book *Applied Security Visualisation,* Marty says that *"a powerful method of showing and highlighting important data in a graph is to compare graphs"* and given that the data from each node was relevant to each other, we decided to show the activity of each CPU side by side in order to allow for the exact same comparison that Marty speaks of. Another principle we borrowed from Marty (2008) was the use of graphs displaying *"normal activity"* in order to immediately see that which differs from the norm. We employed this in two different ways, firstly by having a ten minute period of normal activity before introducing the malware into the environment, but also by testing malware that we knew did not have a method of propagation over a network. Both of these allow the viewer to see "normal activity" and see what activity looks like on a machine that is compromised compared against machines that have not been. Furthermore, screenshots were included at the appropriate spikes in usage on the CPU visualisations simply for some clarity as to what the machines would look like at that given moment, giving an accurate impression as to the networks overall state.

The visualisations we created changed drastically over time, starting simply with the plotted values and progressing to include the X and Y labels, a legend, and where possible in the network visualisations, the appropriate IP address of the activity in its row. These changes were made to increase the overall readability of our visualisations, and while these changes may seem simplistic the overall impact that including such changes is drastic. Unless the purpose of your visualisation is to intentionally have the reader interpret the data in their own way then you should always have these elements. Therefore since our the intent of our visualisations is to inform and not necessarily to be interpreted, the inclusion of these elements is key. Once again, as Marty (2008) says *"Graphs without legends or graphs without axis labels or units are not very useful"*.

This is not to say that the visualisations we have created are entirely perfect however, and whilst we believe they fit into and achieve the goals we established, we also believe that they do this in the simplest way possible. Disregarding possible changes to the data gathering methods, there are plenty ways we can change the way in which the data is visualised in order to better convey our data. This is especially prevalent in the visualisation of the network logs, which whilst interpretable could definitely do with some more fine tuning in order to more accurately convey the disparity between the sending IP addresses and receiving IP addresses. The CPU usage graphs display their data rather perfectly in their current state, what they do suffer from however is issues of scalability, since the visualisations we created only include the four nodes and can still become rather busy at particular times. This method of visualisation would not be appropriate for visualising CPU usage across larger networks that better represent corporate networks, and would almost certainly be unreadable.

Overall we believe that the visualizations can be used in order to teach, inform, and assist in both study and decision making at a basic level, and with some additional research and modifications we can improve the visualisations to contribute at a greater level.

## 6.2 Characteristics

Our second collection of objectives were rather questions to be answered once we had reached this particular stage, all of them to do with particular characteristics or use cases, which we will now discuss in order.

From what we have studied we believe that malware propagation is not technically deterministic, whilst it is almost certain that when a malware searches for machines to propagate to it does it in relation to its own IP address and scans for local variations, this does not mean that the way in which it propagates is deterministic. This is because there are many additional factors that influence the propagation of malware, the most prominent simply being speed. For example, in the case of our WannaCry tests, it is more than likely that once it has infected one machine that it scans for IP addresses similar to its own. With the initial infection occurring on 192.168.0.10, it would be fair to say the next it would scan for is 192.168.0.11, then 192.168.0.12, and so on. But in the test case we visualised, 192.168.0.12 became infected before 192.168.0.11, which as mentioned could be due to something as simple as it managed to receive and accept the SMB connection from the infected machine faster than the others. One could argue that because we can relate certain

activity to the order of propagation that it is deterministic, but there are so many contributing factors to the order of propagation that unless you have an absolute clear view of every single detail that could affect propagation, it is simply going to appear as random or perhaps pseudo-random.

We unfortunately were not able to appropriately study the effects of network structure on propagation for many different reasons, the most notable being that having to configure the structure of our virtual network greatly increased the setup and tear down time of our tests and would stop us from quantitatively studying propagation. The second notable reason is that we believe the structure of a network would be more interesting to study when the creation of larger virtualised networks is possible, and the hardware we worked with in this study was not capable of such a large amount of visualisations.

The final question is if our works can be used to effectively respond to an active threat event, and the answers is rather complex. The system we created to gather our data cannot be used to respond to an active threat event, and our visualisations cannot be either. When we are gathering the data we perform no analysis on it, we merely store it for later analysis and visualisations. Therein also lies the problem with our visualisations, we are visualising that which has already happened, not what is happening. We can certainly use our visualisations to pick characteristics of certain malware, or perhaps create generalisations to assist in future decision making during active threat events. We could instead adapt our system to visualise the data in real time and provide a greater assistance during an active threat, but there would still be a considerable delay in reporting, meaning that one would also have to develop some way of linking particular points of data and training the system to make effective judgements on its own. In short, even when actively visualising live, the delay between visualisation, judgement, and making a decision may incur a great loss already, this is particularly noted in relation to WannaCry which spread across our four machines in just under two and a half minutes.

### 6.3 Further Findings

There were a few additional findings noted during the testing section of this project we believe to be of particular importance, the first being the conflict between qualitative and quantitative study of malware. In our study we employed the method of quantitative study in order to collect data to create a broad impression of propagation across multiple pieces of malware, and it was because of this that we realised that whilst the number of malware that propagate is high, the number of malware that propagate due to a flaw inherent in the Windows 7 operating system is not. Eternalblue functions by exploiting a flaw in the SMB protocol, which is in of itself present in the base Windows 7 operating system, but other malware often spread through methods more akin to compromised email accounts or by stealing peoples email address books and emailing itself to others. From this we feel the need for a distinction between external propagation and internal propagation. External propagation meaning that the malware has to effectively leave the network in order to propagate, even if it is simply to a mailing server, and internal propagation is when a malware does not have to leave the network and can instead spread over the network using exploits in the operating systems present.

Building on this, we made particular note of a weakness within our own system, this being that we are incapable of testing malware that utilises any form of external connection in order to function due to our virtual networks lack of internet connection. The first way this occurs is aforementioned, if the way in which a malware propagates is using compromised emailing accounts or lists, then we cannot visualise it effectively because the systems cannot send emails due to the inability to contact mailing servers. The second being the case of CryptoLocker, if a malware has to communicate with a command and control server in order to operate, it cannot, and therefore will not function. The third and final being the case of Emotet, or more broadly: dropper type malware, where after deploying its own payload it downloads additional payloads in order to increase functionality, but it obviously cannot. In the particular case of Emotet is was stopped from downloading Trickbot, which would allow it to exploit the SMB vulnerability and therefore propagate.

Another observation once again comes back around to our point of qualitative versus quantitative study, there are many malware which function or propagate due to exploits in particular programs rather than exploits in the operating system, and since our testing environment was standardised to allow for quantitative study it is entirely possible that

these potentially very interesting cases were or would be entirely missed. Therefore one could argue that a project such as this could require a longer time span in which to be conducted, so that the testing approach can be adapted to allow for such malware to be accommodated for.

Our final observation comes from a possibility considered at the beginning of this study, and made more apparent after testing Locky. This is the concept of contextual awareness, whether malware is able to detect if it is running in a virtual environment or not. This presents two problems, the first being a weakness of our system, which stems from the effort to streamline the testing by rolling the machines to a clean state after each test. The problem here being that the clean state is very minimalistic and hasnt had the chance to accrue various logs like a long ran system might have, similar to what Miramirkhani et al (2017) discuss in their paper regarding artifacts, our system may have had time to gather what Miramirkhani et al (2017) refer to as “Indirect Artifacts” but not any “Direct Artifacts” which stem more from human interaction. In reality this could be overcome by testing on machines that have actually been used for real life circumstances, but Miramirkhani et al (2017) also conclude this, and acknowledge that this would present certain ethical complications. The second complication comes from the very way in which we create our virtual machines and networks using the program VirtualBox. VirtualBox is extremely useful and allows for the quick creation of virtual machines and networks for testing malware, but more modern evasion techniques are beginning to learn how to look for particular registry entries that are unique to machines that are running through VirtualBox. Or more obviously, the malware can simply look for whether or not the machine has Guest Additions installed and evade activation on machines that do, but this particular method we addressed and avoided in our testing. This isn’t a problem unique to VirtualBox either, its competitor VMWare also possesses similar problems where the change isn’t the in problem but the context in which is occurs.

# 7 Conclusions

This section will surmise what has been achieved within this project, looking at the program created, the data gathered, and the visualisations created in brief. It will then conclude with plans for future work.

## 7.1 Evaluation

Much has been produced during the course of this project, including the creation of a dashboard program capable of launching multiple virtual machines through VboxManage. The same dashboard is able to collect metric data from all of the running machines and collate them in a CSV file that can be utilised to visualise the activity of all the machines. The visualisations we created prove to be an excellent tool in observing the metric and network log activity of a threat active on a network, using these visualisations one could gain insight to exactly how activity taking place on and originating from compromised machines looks across an entire network.

Our dashboard program was tested quantitatively on a virtual network that is reminiscent of a home network. With this structure we were able to effectively gather data that allowed for the easy comparison of different machines, without including so many machines that the data became too noisy and therefore hard to interpret. The virtual network we created is fairly notable, since most retroactive malware analysis takes place on virtual machines or networks running Windows XP, the decision to instead use Windows 7 we believe gave us the ability to explore what could be considered rather uncharted areas.

We also believe that our work can be effectively adapted for those also looking to study the area of malware propagation and characteristics, appropriately detailing our setup and providing sufficient proof of the system working both in concept and in practice. We would also of course encourage fellows to use our work to study malware for themselves, and therefore assist in the effort of combating the growing tide of cyber threats we are currently experiencing.

## 7.2 Future Work

Most of the work to be done revolves around the dashboard program, firstly we want to modify the layout and operation of the page to allow for the data to be gathered like it does,

then be visualised as seamlessly as possible on the same webpage. This will allow one to receive a live impression of the activity of machines running on the virtual network based on the metric usage of each of individual machine. This process may require some sweeping modifications to be made, first of all the dashboard will have to be modified to include graphs created through libraries such as d3.js or chart.js. Then the issue of passing the graphing scripts the data will have to be resolved, either passing the data from the Python backend to the HTML frontend using AJAX or using the previously implemented CSV functionality to visualise the entire graphs straight from said CSV.

Following this, we wish to rework the way in which the virtual network is created. The idea is to modify the dashboard to create virtual machines, organise them into groups, and launch said groups of virtual machines. This would greatly reduce the setup and tear down time of the testing phases using the dashboard an create a much more streamlined experience. The dashboard should also be able to modify virtual machine snapshots en masse to allow for the easier rollback of machine states, which would also consequently reduce the tear down time.

Finally, in regards to the data collection, it might be worthwhile to explore additional methods of metric gathering other than just vboxmanage, since whilst vboxmanage provides much utility, it also has some difficulties when it comes to data gathering. Over half of the metrics available to vboxmanage are only useable when VirtualBox Guest Additions has been installed on the virtual machine, but as explored various times in this paper, Guest Additions provides an easier method for contextually aware malware to detect if it is running in a virtual environment or not. Which leads into another part of future work, to further research and develop a setup that is able to effectively negate the more simplistic contextual awareness implementations such as detecting installed programs and observing run time logs, then possibly beginning to look at the more complex and foolproof methods such as registry entries, hardware names, and driver version checking.

We will always be looking at methods of improving our visualisations to allow for the effective communication of our findings, potentially even allowing for multiple options for different types of visualisations in the live version so the user can always have multiple perspectives. Then finally we would want to look at visualising larger, more corporate looking networks to get a greater impression of what an active threat on a large, interconnected network with many different nodes and topologies would look like.

With this much to do, we encourage any fellow researchers to explore this field and to continue to develop ways in which we can battle and study malware.

# 8 References


Afianian, A., Niksefat, S., Sadeghiyan, B. and Baptiste, D. (2019) Malware Dynamic Analysis Evasion Techniques: A Survey. *Acm Computing Surveys* [online]. 52 (6), pp. 126:1-126:28. [Accessed 26 May 2020].

Bai, J., Wang, J. and Zou, G. (2014) A Malware Detection Scheme Based on Mining Format Information. *Scientific World Journal* [online]. 2014 [Accessed 01 June 2020].

Chakkaravarthy, S.S., Sangeetha, D. and Vaidehi, V. (2019) A Survey on Malware Analysis and Mitigation Techniques. *Comouter Science Review* [online]. 32, pp. 1-23. [Accessed 26 May 2020].

Creese, S., Goldsmith, M., Moffat, N., Happa, J. and Agrafiotis, I. (2013) Cybervis: Visualizing the Potential Impact of Cyber Attacks on the Wider Enterprise. Ieee [online]. [Accessed 21 July 2020].

Cuckoo (2010) *Cuckoo Sandbox* (2010) [computer program]. Available from: https://www.cuckoosandbox.org [Accessed 31 July 2020].

Gove, R. and Deason, L. (2018) Visualizing Automatically Detected Periodic Network Activity. *2018 Ieee Symposium on Visualization For Cyber Security* [online]. [Accessed 02 March 2020].

Hosseini, S. and Azgomi, M.A. (2016) A Model For Malware Propagation in Scale-free Networks Based on Rumor Spreading Process. *Computer Networks* [online]. 108, pp. 97-107. [Accessed 01 June 2020].

Marty, R. (2008) *Applied Security Visualization*. United State of America: Addison-Wesley Professional.

Mills, A., Spyridopoulos, T. and Legg, P. (No date) Efficient and Interpretable Real-time Malware Detection Using Random Forest. *(No Place)* [online]. [Accessed 16 February 2020].

Miramirkhani, N., Appini, M.P., Nikiforakis, N. and Polychronakis, M. (2017) Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-tear Artifacts. Ieee Symposium on Security and Privacy [online]. [Accessed 21 July 2020].

National Health Service UK (2018) *Lessons learned review of the WannaCry Ransomware Cyber Attack*. Skipton House: National Health Service UK.

Oracle Corporation (2007) VirtualBox (6.1.10, 2020) [Computer Program]. Available from: https://www.virtualbox.org [Accessed 1 February 2020].

Patel, A. and Tailor, J. (2020) A Malicious Activity Monitoring Mechanism to Detect and Prevent Ransomware. *Computer Fraud and Security* [online]., pp. 14-19. [Accessed 16 February 2020].

Rhode, M., Burnap, P. and Jones, K. (2018) Early Stage Malware Prediction Using Recurrent Neural Networks. *Computers and Security* [online]. 77, pp. 578-594. [Accessed 15 February 2020].

Sharafaldin, I., Lashkari, A.H. and Ghorbani, A.A. (2017) Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *Proceedings of the 4th Internation Conference on Information Systems Security and Privacy* [online]., pp. 108-116. [Accessed 21 July 2020].

Nativ, Y.T., and Shalev, S. (2020) *theZoo*. Available from: https://github.com/ytisf/theZoo [Accessed November 2019].

NTFS123, (2020). *MalwareDatabase*. Available from: https://github.com/NTFS123/MalwareDatabase [Accessed 25 June 2020].

Hispasec Sistemas (2004) *VirusTotal*. Available from: https://www.virustotal.com [Accessed 31 July 2020].

Xiao, F., Lin, Z., Sun, Y. and Ma, Y. (2019) Malware Detection Based on Deep Learning of Behaviour Graphs. *Mathematical Problems in Engineering* [online]. 2019 [Accessed 01 June 2020].

Yu, S., Gu, G., Barnawi, A., Guo, S. and Stojmenovic, I. (2015) Malware Propagation in Large-scale Networks. *Ieee Transactions on Knowledge and Data Engineering* [online]. 27 (1), pp. 170-179. [Accessed 14 February 2020].

Zhuo, W and Nadjin, Y (2012) MalwareVis: Entity-based Visualization of Malware Network Traces. *Acm International Conference Proceeding Series* [online]., pp. 41-47. [Accessed 15 February 2020].

# 9 Appendices

## Appendix 1 - WannaCry Reinfection Close Ups