Developing a contained and user emulated malware assessment platform

F. Potter*, S. Hodzelmans*

Supervisors: V. Van Mieghem †, H. Hambartsumyan†

*Security and Network Engineering,
University of Amsterdam,
Email: {fpotter,shodzelmans}@os3.nl

†Deloitte,
Email: {vvanmieghem,hhambartsumyan}@deloitte.nl

Abstract—Penetration testers and red teams develop malware to simulate real digital threats to organizations. In order to test which virus scanners detect their malware before using it, they would like to test their malware, without risking sample submission to the AV vendors. Furthermore, they want to test their malware as if it was executed by a user in a realistic way and within a controlled environment. In this research, we investigate the kind of traffic that AV software generates, how sample submission can be blocked and how the user behavior can be emulated.

The traffic analysis showed various kinds of traffic, however sample submission wasn't one of them. Since we didn't observe any sample submission, we can only speculate on the best approach to block this traffic. Based on what we did observe, we recommend a whitelisting approach. When applying user emulation and direct scanning we observed that the static analysis resulted in a higher detection rate. However, the false positive rate was also higher. The dynamic analysis with user emulation on the other hand has a lower detection rate, but doesn't have any false positives. Another interesting observation was that in some cases a difference in on- or offline scanning can occur.

We conclude that triggering sample submission isn't trivial, but suspect that whitelisting would be the best approach to prevent it. We also showed that dynamic analysis can be automated using user emulation and adds value besides static analysis.

Index Terms—Antivirus, traffic inspection, user emulation, malware testing, red teaming

1 Introduction

Nowadays malware is the main source of IT security threats [1]. Michalopoulus et al. [1] state that antivirus (AV) software provides protection against malware, and therefore is an important defense factor. In order to allow users and organizations to check suspicious content, several online malware analysis platforms (e.g. VirusTotal, Metascan, Camal, Malwr and AVCaesar) have been developed [2]. These malware analysis platforms provide the option to upload a file and scan it by multiple AV vendors. This gives the advantage of being able to have a majority vote, whether the file is malicious or not. However, these malware platforms are not on-premise and therefore the uploader of the file isn't in control over where the uploaded file goes or who sees the data [2]. This can be a problem if the uploaded file contains private data.

According to Debrie et al. [2] the Incident Response and Malware Analysis (IRMA) platform provides an on-premise, open-source and automated malware analysis service. IRMA allows the uploaded file to be contained within the network of the organization, and therefore to stay in control. However, one cannot be certain that the AV software doesn't do sample submission. For example, Kaspersky accidentally ended up with confidential NSA files [3]. These confidential files were part of a zipped folder, that also had malicious binaries, and therefore was submitted to Kaspersky.

Debrie et al. [2] state that IRMA can detect if a file is malware but doesn't detect what the malware would do if the user executes it. Furthermore, IRMA allows the user to be in control over the file, but doesn't prevent the AV software to learn the signature of the malicious content [2]. Testing which AV software detects the malware is useful for red teams and penetration testers, because their malware doesn't harm the organization, but shows how vulnerabilities can be exploited by malicious parties in order to do harm to the organization. It is also usually the first step in a simulated attack by a red team, and thus critical for the rest of the attack path. For these reasons, red teams and penetration testers want to test whether their malware will be detected, without the AV software uploading a sample to the vendor. When sample submission occurs, AV vendors will learn about the malware and add it to their database, ensuring that the malware gets detected. When the malware created by the red team gets detected by the AV software, it becomes useless for them and severely disrupts the simulated attack.

2 Previous work

2.1 Red teaming and penetration testing

Randhawa et al. [4] state that "everyday people and organizations are more dependent on cyber systems". They also state that "the loss, degradation, corruption or unauthorized access and exploitation of critical business applications can have significant impact on the cyber dependent business and therefore represent a threat to business objectives". In order to address these threats red teaming and penetration testing can be used to detect security weaknesses and the impact that those weaknesses have. Red teaming and penetration testing also helps organizations to exercise incident response mechanisms. According to Randhawa et al. [4] penetration testing provides security insight into the network and system vulnerabilities as a whole, while red teaming is an attack simulation in order to test the organization's detection and response capabilities when under attack.

2.2 Incident Response and Malware Analysis

IRMA provides an on-premise, open-source and automated malware analysis platform, that our research will be advancing on. Figure 1 shows how the IRMA platform is split in a three tier model, consisting of a frontend, brain and probes [2]. The frontend handles the user interaction and uploading of the file. When a file is uploaded and a scan is launched, the frontend first checks if the file has been uploaded before and therefore already has scan results for this file in the database (i.e. caching). If the file doesn't exactly match with the results from the database or a rescan is required, it stores the file on the file server of the brain. Next, the brain schedules the analysis of the file in a queue (i.e. asynchronous), so if one of the probes is ready to analyze the file, it pulls it from the queue. Once the probe has processed the scanning of the file, it returns the result back to the brain. The brain then forwards the result back to the frontend, giving the user the information (s)he requested [2].

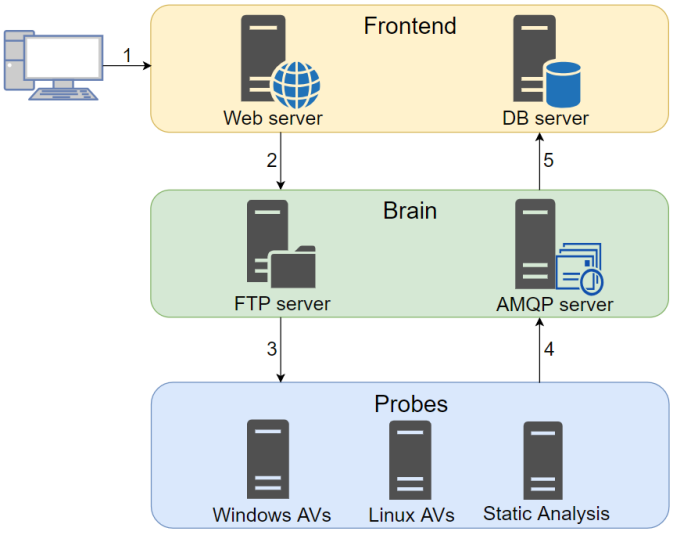

Fig. 1. IRMA three tier model

2.3 User emulation

IRMA provides static analysis by checking signatures of files and compares them with known signatures of malware [2]. According to Sanok [5], static analysis provides on-the-fly scanning and offers high speed and performance. However, with static analysis, malware can still remain undetected. Carvey [6] states that 'static analysis of executable files has a number of limitations'. One of those limitations is that one will not know what the executable exactly does without launching it. Therefore, using dynamic analysis, which uses heuristics, is a more sophisticated solution. Sanok [5] states that heuristics is a method to determine the probability that a file or program is malware by analyzing its characteristics. These characteristics are based on the user's actions, file origin, execution method, parent process, etc. In order to see the heuristics that trigger AV software to block malware, user emulation will be needed.

User emulation is the use of automated small GUI-based workflows, such as opening an email or downloading a file with a web browser and executing it. There are several options to achieve this task, such as Sikuli or Selenium for web applications. Other options based on the Python language are pywinauto or pyautogui. User emulation could be interesting for red teams and penetration testers in order to see which characteristics trigger the AV software and how this could be avoided.

2.4 Antivirus software analysis

There has been little investigation into how AV software operates, and as far as we could find, no one has investigated the traffic generated by AV software (at least, no one has published about it). Radvilavicius et al. [7] have made an overview of how different AV products do their real-time scanning. Al-Saleh and Crandall [8] gave a detailed explanation of how Clam-AV, an open source virus scanner, does the scanning. They also showed that it can be attacked to determine the age of the signature database of an AV engine. Eronen et al. [9] and Xue [10] showed that a multitude of attacks are possible on AV programs.

3 Research question

With IRMA providing an on-premise solution for malware analysis, we want to examine the traffic generated by AV software to prevent malware sample submission and create a test environment for red teams and penetration testers. Furthermore, we want to emulate browsing user behavior to allow on-access scanning of malware. Therefore, we conducted the following research question:

'How can malware be tested for detection of antivirus software by emulating user actions, without the AV vendor learning about the malware?'

To provide an answer to this research question, the following three sub-questions have to be answered:

- What traffic is generated by AV software?
- How to prevent AV software from notifying and submitting the red team's malware to the AV vendor?
- Are there any differences between direct scanning and user emulated detection rates?

4 METHODOLOGY

In order to answer our first sub question 'What traffic is generated by AV software?', we will select the top three most used AV vendors and test what traffic they will generate under the same circumstances (i.e. updates, quick scan, full scan, detecting malware). Computer Profile [11] has done a market share survey in Belgium and found that those are McAfee (27%), Symantec (25%) and Trend Micro (17%). Therefore, we will focus our research on the AV vendors Symantec (version 22.16.3.21), McAfee (version 16.0 R14) and Trend Micro (version 15.0.1212). During the research we decided to also add Kaspersky (version 19.0.0.1088 (d)) in the investigation of the observed network traffic part of the study. This was because we couldn't find any sample submission with the other AV vendors, but it is confirmed that Kaspersky does submit samples [3].

In order to see the traffic the AV software generates, we will do the following tests for each AV vendor:

- Performing an update of the AV software (twice)
- Performing a quick scan (twice)
- Running a full system scan (once)
- Scanning five malware samples
- Scanning several samples provided by Deloitte

The updates and quick scan will be done twice, to see if similar traffic is generated when the AV is already up to date or when a file has already been scanned before. The full scan is only done once, since it will be quite time consuming. The five malware samples that will be scanned to see if sample submission is occurring, are from Das Malwerk [12]. The samples of Deloitte consist of a default Cobalt Strike beacon, an obfuscated beacon, and a binary generated by MSFvenom (which is part of the Metasploit framework). A list of the files used can be found in appendix A.

4.1 Test environment AV traffic

The test environment of the AV traffic generation is shown in figure 2. The test environment will consist of a Virtual Machine (VM) running the latest 64 bit Windows 10 as the client and will contain the AV software. Another VM running Ubuntu 18.04 (latest Long Term Support) with mitmproxy [13] will be used to intercept the traffic. For each AV vendor a separate Windows VM will be created, and only one VM is run at a time to ensure traffic belongs to a certain AV product. Furthermore, the Windows VM has one internal network adapter that is connected to the Ubuntu VM. The Ubuntu VM also has a second external network adapter to provide network access through Network Address Translation (NAT).

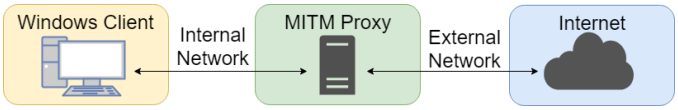

Fig. 2. Man-In-The-Middle Proxy test environment

Oppliger et al. [14] state that 'most e-commerce applications are using Secure Socket Layer (SSL) or Transport Layer Security (TLS) to protect the communication channel between the client and server'. However, a Man-In-The-Middle (MITM) attack can be used under specific conditions to circumvent the protection that SSL/TLS adds to provide end-to-end security. This can be done by placing a third host between the client and server and act as a relay on behalf of the client, in our case mitmproxy. Secondly the root certificate of mitmproxy has to be imported in the trusted certificate store of the Windows VM, so that the client trusts it as a relay [13].

Figure 3 shows how mitmproxy intercepts web traffic and redirects it, given that the root certificate of mitmproxy is added to the trusted certificate store. Since the client receives the proxy certificate (certificate P) instead of the server certificate (certificate S), the proxy can sniff all the web traffic.

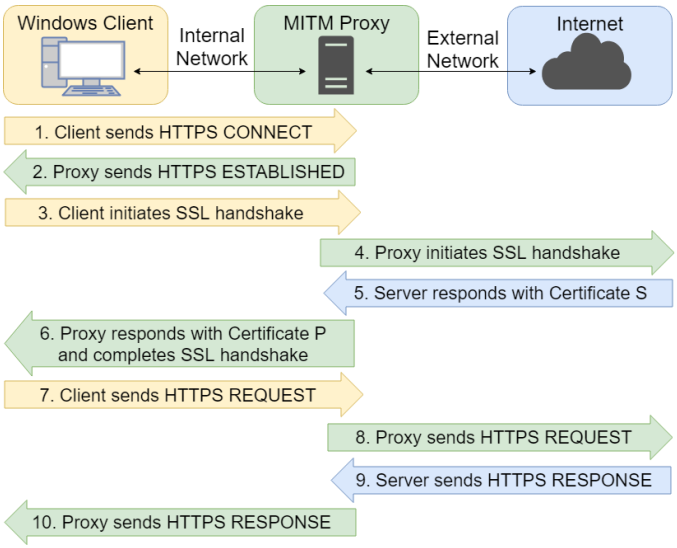

Fig. 3. mitmproxy

According to the National Cyber Security Centre (NCSC) [15] a tool such as mitmproxy could prevent applications, e.g. AV software, from connection with the server. This happens when a client only trusts a specific certificate from a server (i.e. certificate pinning). However, when a client is under full control, it is possible to bypass certificate pinning [13].

According to Evans et al. [16] certificate pinning has been developed to avoid MITM attacks. Certificate pinning is done by storing a copy of the certificate or the fingerprint of the certificate in the client application. Whenever the client initiates the SSL handshake with certificate pinning, it performs two steps. The first step is checking whether the server certificate is issued by a trusted Certificate Authority (CA). The second step is added by certificate pinning, and verifies if the certificate of the server or a certificate higher in the chain (i.e. one of the issuer's certificates) matches with the certificate or the fingerprint of the certificate stored in the client application. If both steps were successful, the SSL session is established. However, if the checking of the server certificate or the pinning process fails, the session will not be established.

Certificate pinning brings the risk of service unavailability (i.e. when the pinning process fails). The Fraunhofer Institute for Communication, Information Processing and

Ergonomics (FKIE) [17] states that 'organizations make use of middleboxes or MITM proxies to intercept traffic'. Because of this, we assume that certificate pinning will not be used by the AV vendors. However, if the AV software uses certificate pinning, mitmproxy can't negotiate its own certificate during the SSL handshake. Meaning we can't intercept the AV traffic with the common mitmproxy test environment and need to find other solutions to intercept the traffic. However, if we are able to intercept traffic, we will analyze it with Wireshark. The intercepted traffic will also be aggregated using scripts. The aggregation will be done for each vendor, and then these sets will be aggregated in one list. We will count in how many sets a given endpoint has occurred, in order to differentiate the traffic generated by the AV software and the traffic generated by e.g. Windows or Microsoft Edge.

4.2 Preventing notification and submission of red team's malware

In order to answer the second sub question 'How to prevent AV software from notifying and submitting the red team's malware to the AV vendor?', we will look into three possible methods. These three methods, for preventing specific traffic generated by AV software, are:

- Blacklisting the undesired traffic
- Whitelisting the desired traffic
- After the AV software is up-to-date, taking the Windows VM offline

According to Townsend [18] blacklisting is blocking known bad or unacceptable behavior. If one specifies which AV traffic is unwanted, one can block this traffic from traversing the network. Townsend states that 'a primary advantage of blacklisting is that it is conceptually simple to recognize a few bad things, stop them, and allow everything else'. Furthermore, blacklisting eases administration, since maintaining blacklists can be delegated. Therefore, blacklisting the undesired traffic is the least drastic method.

Whitelisting means allowing the traffic that should occur is known good (or at least acceptable) [19]. According to Townsend [18] whitelisting is better security wise, since it avoids security vulnerabilities that a blacklist doesn't block. Whitelisting prevents that traffic in the first place, because it wasn't necessary traffic. However, in order to ensure all desired traffic is allowed, the traffic needs to be analyzed regularly, to assure that no changes occurred that could make a service unavailable. If changes occur that are necessary for the AV software to run properly due to e.g. an update, one doesn't want the service to become unavailable. Therefore, whitelisting is administrative more intensive than blacklisting.

Taking the Windows VM offline is the most drastic option, since it will prevent any traffic to and from the internet. However, this poses a problem for long-term usage. When the VM or AV software have to be updated, they have to be brought back online. If the AV software holds the malware in a queue of sorts, there is the risk that it gets uploaded the moment the VM regains internet access. However, this can be resolved by snapshotting a VM and restoring it back to a moment before samples were scanned. According to Cohen and Nissim [20] snapshotting is a virtualization technique that allows the administrator to store the current state (take a snapshot) of a (running) VM. Furthermore, they state it can be used to restore a certain state before a process takes place, in our case scanning malware samples. 'In addition, the snapshotting process is fully trusted, because a malware cannot manipulate it since the machine is suspended' [20]. To test whether the approach of taking the VM offline during the tests is viable, we will perform the five malware scanning tests of Das Malwerk [12] and the samples of Deloitte both while being offline as well as online.

4.3 Static versus dynamic analysis

The third and last sub question 'Are there any differences between direct scanning and user emulated detection rates?' will be answered by comparing the detection rates of static versus dynamic malware analysis. Since we want to test multiple malware samples to make a comparison of the detection rates in an automated way, we will emulate user behavior to execute malicious files. However, when one emulates user behavior, it needs to have the same results as when it is executed by a real world user. According to Morales et al. [21] a Malware infection Tree (MiT) can help dynamic malware analysis by creating a structure of the processes and files that are related to the malware. Therefore, we will use the MiT in order to verify if user emulation is identical to real world user execution when it comes to dynamic malware analysis. For the user emulation we will use pywinauto and pyautogui, which are two Python libraries to emulate the Windows Graphical User Interface (GUI). Pywinauto can emulate user behavior by using the Windows Accessibility Application Programming Interfaces (APIs) to communicate with the GUI. Pyautogui uses a pure Python script to emulate GUI actions and can use screenshots to recognize when and where to perform a certain action (e.g. to click 'yes' when a pop up screen appears).

In order to collect the detection rates, we will test 19 true positive and 19 false positive malware samples (see appendix B). These will be tested in a contained Windows VM, to avoid a production environment from getting infected. To be as realistic as possible, we will use the same setup as described before with the mitmproxy (figure 2), which allows us to see the traffic conducted during the static and dynamic malware analysis. Furthermore, during the dynamic analysis, malware has to be executed in order to see the MiT tree. Pandey et al. [22] describe the Process Explorer as an advanced task manager that gives insight into the processes running on the system, including its Dynamic-Link Libraries (DLLs), handles, events and threads. They specify the Process Monitor as a dynamic analysis tool to view real-time process and registry activity. Therefore, we will use the Process Explorer and Process Monitor tools in order to verify the MiT tree of user emulation to real world user behavior.

5 RESULTS

5.1 AV traffic generated

While performing the update, quick scan, full scan and download of five malware samples on the system, all the AVs generated a lot of traffic. However, we were unable to find any sample submission done by the AV vendors. The traffic that was generated was in most cases (partially) hashed or encoded. Besides the hashes we also saw a lot of calls to analytic endpoints, which were collecting information about the Operating System (OS) and the hardware of the VM. The complete list of the endpoints observed can be found in appendix C. Even though we couldn't find proof of sample submission, the following was observed for each AV vendor.

McAfee sends hashes of the files it scans, along with some metadata of the files such as the signature. It also sends various analytic data periodically, and uses Google Analytics besides its own analytics endpoint. It detected all the Deloitte provided samples except the obfuscated beacon. From Das Malwerk it only recognized the first two (starting with 1e84 and 1f7b) samples.

Symantec sends a reputation request when it scans a file, containing the name of the file, the source and some additional info of which we are not sure what it exactly is. Furthermore, we also see other requests which POST more comprehensive data, when we download the malware samples directly from the internet and real time scanning is enabled. Among this data we observed the MD5 and SHA256 hash of the Microsoft Edge binary (which we used to download our malware samples). We also saw the base64 encoded Uniform Resource Locator (URL) of the download and a "data buffer" field containing about 800 bytes. According to Symantec [23] and based on the endpoint this is a "ping submission", which is used to combat false positives. What we observed was that the download wasn't even allowed to complete in these cases. Furthermore, it detected all the samples except the obfuscated beacon.

Trend Micro sends for each sample a GET request over an unsecured HTTP connection with a very long URL. The first part of the URLs is the same for all these requests, while later parts of the URL differ. The URLs also vary in length. Also of note is the fact that during the update process Trend Micro uses HTTPS, but doesn't provide a Server Name Indication (SNI). This could be a potential weak point in the communication, if the client doesn't do strict checking on the certificate.

Trend Micro was the only tested software which showed a difference in whether we tested online or offline. The difference is in the first Das Malwerk sample (1e84): it wasn't detected when the scan was run offline, but it was detected when it was run online. The rest of the samples showed no difference. Whether on- or offline, Trend Micro detected all the other Das Malwerk samples and the MSFvenom sample of the Deloitte samples. The other samples that Deloitte provided were not recognized.

Lastly, while examining Kaspersky we didn't see any traffic that we could relate back to the samples we tested. We did however see a lot of 400 (Bad Request) and 502 (Bad Gateway) HTTP errors during the scanning. The error messages seem to indicate that the client tries to speak plain HTTP over port 443 and therefore the connection gets closed. Also of note is that Kaspersky does use certificate pinning, but only does so during the registration. The certificate that is pinned isn't signed by a trusted CA. However, Kaspersky intercepts the SSL traffic, and for that purpose inserts a generated root certificate in the Windows trust store. Kaspersky is the only tested AV to do does so. It detects all the samples, except the obfuscated beacon. It did detect the first (1e84) as possibly malicious, but left it up to the user to determine what action should be taken.

All of the above traffic took place over port 80 and port 443 using the HTTP or HTTPS protocols. We didn't observe any other traffic over other ports connecting to the internet, except for DNS requests.

5.2 Preventing malware submission

Since we haven't been able to observe or generate any sample submission, we have no concrete results on how to prevent it. Therefore, we couldn't test whether a solution to prevent sample submission is preferred over another. However, based on the observed behavior we do think that a whitelisting approach would fit best. First of all whitelisting would allow the VM and AV software to be kept up to date. Secondly, most AV software expects to be online. As stated before, Trend Micro performs less when offline in comparison to when it is online. Furthermore, Symantec gives a warning when running a scan while offline. These two facts make the option of taking the VM offline entirely less desirable. Blacklisting is also less desirable. First of all, one needs to know what to block in advance, and as our research showed, this is not trivial. Secondly, if the endpoints were to change, e.g. because of an update of the AV software, the blacklist needs to be updated as well. A possible consequence could be that sample submission is not blacklisted anymore, and the sample will be submitted. The reverse is true for whitelisting: if allowed endpoints changed, the whitelist needs to be updated. In this case, the sample will not be submitted, but there is a risk that the malware analysis or update process will not function correctly. However, whitelisting generally requires more work than blacklisting, especially when first creating the whitelist. The above leads us to believe that whitelisting is more desirable for red teams, since sample submission can severely disrupt a red team operation.

The approach to whitelisting should be based on hostnames, traffic size and direction, and possibly content. Filtering on IP addresses will be hard, because nowadays these are often shared, especially in the cloud. Filtering on hostnames will be a benefit, since we observed that all vendors split different components/services/aspects of AV software (such as registration, updates and signature checks) over different hostnames. Filtering based on the size and direction of traffic could be effective if the sample is not submitted in several small chunks. This has to be combined with the direction of traffic, because updates are significantly larger then the malware samples. Finally, filtering on content would depend heavily on how a sample is submitted. If it is submitted in whole, unencoded, over a plain connection it would be a guaranteed way to prevent submission. However, from what we observed this

is probably not the case. Vendors mostly use encrypted connections and often also encode the data they send in some way. Looking in the encrypted connections can be solved by using SSL interception as we did in our research, but decoding the data on the fly and checking that against the sample would probably require a custom solution, and would also require to understand and know exactly how the AV software encoded the sample. Furthermore, if the upload is done in parts or only a part of the entire malware file is uploaded, matching and detecting this becomes significantly harder. Therefore we think that whitelisting on content should only be considered if the other methods are deemed insufficient to separate the desired traffic (such as software updates) from the undesired traffic (such as sample submissions).

5.3 Static versus dynamic analysis

In our setup we compared the AV vendors Symantec, Trend Micro and McAfee based on static and dynamic analysis of 19 true positive and 19 false positive samples. True positives are malware samples that are harmful and false positives are samples that aren't malware and therefore harmless. The table of the samples and the detailed results can be found in appendix B. The static analysis was done by direct scanning and resulted in the graph shown in figure 4. The static malware analysis shows that Symantec detected all true positives malware samples, while Trend Micro detected 89% and McAfee detected 52.6% of the true positive samples. Furthermore it shows that Symantec has the highest false positive rate (47.4%), followed by Trend Micro (37%) and McAfee (26.3%).

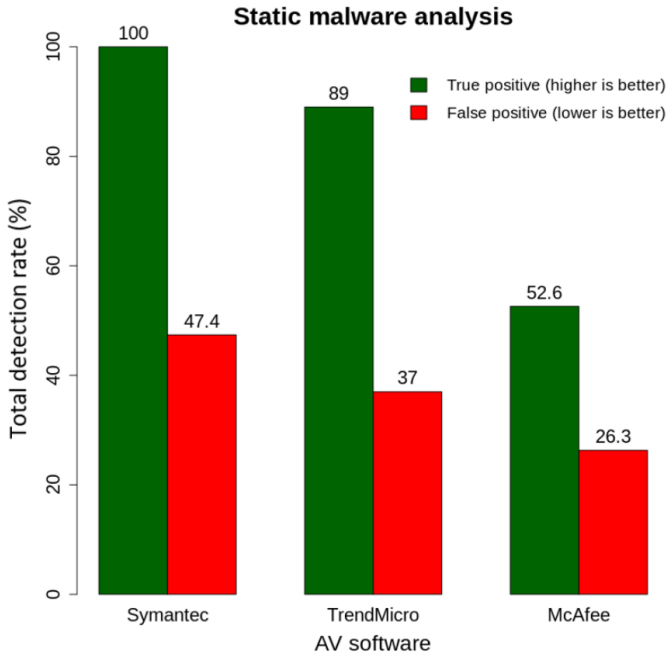

Fig. 4. Static malware analysis

The dynamic analysis was done by emulating the downloading of malware samples through the Microsoft Edge web browser. Before the download was started, real-time scanning had to be turned off, to disable signature scanning (i.e. static analysis) and allow on-access scanning of samples. When the malware samples were downloaded, the execution of the malware was emulated. Then real-time scanning was turned back on and we gave the AV software 5 minutes, to see if the malware sample was detected by the AV software.

The user emulation of malware samples was done with both pywinauto and pyautogui. We created two Python scripts, one based on pywinauto and the other on pyautogui. These can be found in appendix D. Pywinauto uses a Windows accessibility API that can interact with the GUI. Our pywinauto scripts starts the Microsoft Edge web browser from the desktop. After the web browser has opened, it loads the web page of the web server, that contains the malware samples. Then it selects the malware sample to download (based on the file name given). When the malware sample has been downloaded, the script executes the sample. For some reason pywinauto was very error prone, which resulted in having the user emulation script to be run multiple times with the samples it missed. An example of an error is the web page with the malware samples not being loaded.

Pyautogui on the other hand has to import a Python library to open the web browser and load the web page. Then it goes to a sample to download by using x and y coordinates, rather than a file name as done by pywinauto, and downloads it. Furthermore, pyautogui uses a lot of screenshots to coordinate through the downloading and execution of the malware samples. This makes it more static to emulate user behavior. However, when compared to pywinauto, it is less error prone.

Before the results of the dynamic malware could be used, we had to verify if the user emulation was done correctly. Therefore we checked a subset of four malware samples, two which were true positives and two that were false positives. The four samples tested were:

- 21f5d45c-414b-11e8-bfe9-80e65024849a.file
- Win32.WannaPeace.exe
- netsky4.exe
- newstar.exe

Figure 5 shows the process name, Process ID (PID), description and company name of the four samples run manual, while figure 6 shows the same fields but then when run by user emulation. Apart from the PID, no difference is seen in the process tree of the malware samples when using the tool Process Explorer.

Win32.WannaPeace.exe	244		
newstar.exe	3276		
conhost.exe	5296	Console Window Host	Microsoft Corporation
netsky4.exe	7184		
conhost.exe	4500	Console Window Host	Microsoft Corporation
cmd.exe	5356	Windows Command Processor	Microsoft Corporation
conhost.exe	5808	Console Window Host	Microsoft Corporation
DePass_Micro.exe	308	DePass Micro	Kalyuk Vitaliy (KVSoft Ukraine)

Fig. 5. Process Explorer manual

Win32.WannaPeace.exe	4696		
newstar.exe	2856		
conhost.exe	4872	Console Window Host	Microsoft Corporation
netsky4.exe	3448		
conhost.exe	5136	Console Window Host	Microsoft Corporation
cmd.exe	5536	Windows Command Processor	Microsoft Corporation
conhost.exe	2336	Console Window Host	Microsoft Corporation
DePass_Micro.exe	6908	DePass Micro	Kalyuk Vitaliy (KVSoft Ukraine)

Fig. 6. Process Explorer user emulation

Figure 7 and figure 8 show the time, process name, PID, operation, path and result of the start of the sample Win32.WannaPeace.exe for manual and user emulated execution. These results were captured with the tool Process Monitor and show that besides the PID the results are the same.

The tools Process Explorer and Process Monitor didn't show any difference between the MiT tree of manual behavior and emulated user behavior for both pywinauto as pyautogui. Therefore, the results from the dynamic malware analysis seem to be valid.

Time ...	Process Name	PID	Operation	Path	Result
16:39:...	Win32.WannaPeace....	3936	Process Start		SUCCESS
16:39:...	Win32.WannaPeace....	3936	Thread Create		SUCCESS
16:39:...	Win32.WannaPeace....	3936	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	ReadFile	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	ReadFile	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
16:39:...	Win32.WannaPeace....	3936	Load Image	C:\Users\RP1\Downloads\...	SUCCESS

Fig. 7. Process Monitor manual

Time ...	Process Name	PID	Operation	Path	Result
13:37:...	Win32.WannaPeace....	6776	Process Start		SUCCESS
13:37:...	Win32.WannaPeace....	6776	Thread Create		SUCCESS
13:37:...	Win32.WannaPeace....	6776	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	ReadFile	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	ReadFile	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	QueryStandardInformation...	C:\ProgramData\Norton\{0...	SUCCESS
13:37:...	Win32.WannaPeace....	6776	Load Image	C:\Users\RP1\Downloads\...	SUCCESS

Fig. 8. Process Monitor user emulation

The dynamic malware analysis based on 19 true positives and 19 false positives, shown in figure 9, shows that the false positive rate of the three tested AV vendors are zero, and therefore don't detect harmless samples. However, Symantec and Trend Micro detected most (78.9%) of the true positive malware samples. McAfee on the other hand only had a true positive detection rate of 15.8%.

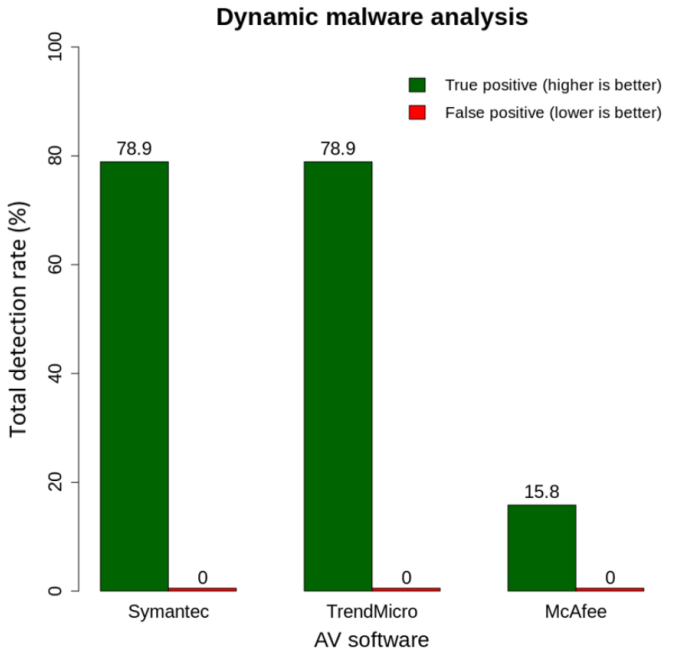

Fig. 9. Dynamic malware analysis

When comparing the static versus dynamic analysis we see the true positive detection rates of static analysis for each AV vendor being higher (better) than the dynamic analysis. However, the false positive detection rates are much better with dynamic analysis (lower is better) in comparison to static analysis.

6 Discussion

During our research we were unable to observe sample submission. A possible explanation can be found on the website of Symantec [23]. They state on their website: 'If a client gets a detection, the client queries Symantec to see if a sample is needed (that is, no formal definition created for this item yet). If a sample is not needed because a formal definition is already created, the client will not submit the sample'. Unfortunately, as the examined AV programs are all closed source, it is hard to know what exactly triggers a sample upload. Testing unknown malware is not a guarantee, as our research showed.

Another possible explanation for not observing sample submission, could be due to the test environment being virtualized. The malware or AV software may work differently when in a virtualized environment in comparison to a physical environment with dedicated hardware. However, the samples provided by Deloitte will run regardless of which environment they are on.

Also, our packet captures were quite contaminated by many calls which were made by Windows and when we

opened the Microsoft Edge browser, which loads bing.com and msn.com addresses by default. We could have taken better care to prepare the VMs to minimize this contamination. Furthermore, we could have run the VM without doing our tests and without an AV scanner installed, to get a baseline of the traffic that is generated by Windows.

A few other minor issues with our method has to do with the use of mitmproxy. First of all, we saw that some connections were blocked because mitmproxy didn't trust the certificate the server provided. In the case of Trend Micro we turned the verification done by mitmproxy off, but in hindsight we should have done this for all captures, so no requests would have been blocked. Secondly, it would have been interesting to test whether the errors which we observed while testing Kaspersky wouldn't occur if we disabled mitmproxy.

The answer we provided to the second research question leaves much to be desired. However, as we couldn't find any way to trigger or even prove sample submission, we could either have dropped that part of the research entirely or do some speculation based on our observations and statements from the AV software. We decided for the latter, since we feel that what we did observe could be used as a basis for further investigation and a possible starting point for a solution.

For the dynamic malware analysis, it is not clear why McAfee has a much lower true positive detection rate than Symantec or Trend Micro. We could have tested more malware samples to get a better indication of this occurrence. However, we tested the malware analysis on two different physical machines and even tested the same samples multiple times to see if it was a coincidence, which didn't give different results. Also without the user emulation, the results of McAfee were the same. What is curious however is that some malware samples from Das Malwerk were not detected by our local McAfee installation, but have been detected by McAfee according to VirusTotal. A link stating these files are detected by McAfee can be found with the files from Das Malwerk [12].

7 Conclusion

Our research showed that AV scanners make network calls for a variety of reasons: installation, registration, updating and checking files for signatures. The fact that we weren't able to trigger a sample submission makes it hard to take countermeasures to prevent sample submission, and shows that this isn't a trivial task. However, if one would want to prevent sample submission as a proactive measure, whitelisting seems to be the most fitting solution. Whitelisting prevents all undesired traffic, even if it is unknown, while allowing the AV software to update regularly. However, an update of the AV software may result in different traffic that needs to be whitelisted. If the whitelist is not updated accordingly, the AV software's performance could be degraded. Whitelisting will require more effort, but is more reliable then blacklisting when it comes to sample submission. Preventing sample submission is important for a red team, since it could severely disrupt a simulated attack. Furthermore, our research showed that dynamic analysis can detect if a sample is a false positive more accurately than static analysis and that user emulation could be used to automate dynamic analysis.

To answer the main question *'How can malware be tested for detection of antivirus software by emulating user actions, without the AV vendor learning about the malware?'* we conclude that user actions can be emulated in multiple ways and have added value over static scanning, with a whitelisting approach to prevent sample submission.

8 Future work

According to our literature research, the AV generated traffic hasn't been widely investigated. It is our opinion that a deep, exploratory investigation in what kinds of traffic AV software exactly generates would be of value. From what we have observed already, and the amount of data we haven't been able to decode or determine what exactly is sent, this would probably yield very interesting results. Also other approaches to investigate what AV products actually do, e.g. reverse engineering, could yield interesting results.

Other future work that can be done in order to advance on our research, is that one could integrate whitelisting to proactively prevent sample submission in an existing on-premise, open-source and automated malware analysis platform such as IRMA.

Finally, the creation of a mechanism to monitor when AV software detects malware could be an area for future work. This would allow user behavior emulation to be used in automated solutions such as IRMA.

9 Acknowledgements

We would like to thank Vincent Van Mieghem and Henri Hambartsumyan of Deloitte for the supervision and support of this project and providing us with samples of malware as used by red teams.

References

[1] P. Michalopoulos, V. Ieronymakis, M.T. Khan, and D. Serpanos. "An Open Source, Extensible Malware Analysis Platform". In: *MATEC Web of Conferences*. 2018.

[2] G. Debrie, F. Lone-Sang, and A. Quint. "IRMA – An Open Source Platform for Incident Response and Malware Analysis". In: *Hack in the Box Security Conference*. 2014.

[3] D. Goodin. *Kaspersky: Yes, we obtained NSA secrets. No, we didn't help steal them — Ars Technica*. 2017. URL: https://arstechnica.com/information-technology/2017/11/kaspersky-yes-we-obtained-nsa-secrets-no-we-didnt-help-steal-them/ (visited on 01/10/2019).

[4] S. Randhawa, Dr B. Turnbull, J. Yuen, and J. Dean. "Mission-Centric Automated Cyber Red Teaming". In: *13th International Conference on Availability, Reliability and Security*. 2018.

[5] D. J. Sanok Jr. "An analysis of how antivirus methodologies are utilized in protecting computers from malicious code". In: *Proceedings of the 2nd annual conference on Information security curriculum development, InfoSecCD '05* (2005), pp. 142–144. DOI: 10.1145/1107622.1107655. URL: http://dl.acm.org/citation.cfm?id=1107655.

[6] Harlan Carvey. "Malware analysis for windows administrators". In: *Digital Investigation* 2.1 (2005), pp. 19–22. ISSN: 17422876. DOI: 10.1016/j.diin.2005.01.006.

[7] L. Radvilavicius, L. Marozas, and A. Cenys. "Overview of Real - Time Antivirus Scanning Engines". In: *Journal of Engineering Science and Technology Review* 5.1 (Mar. 2012), pp. 63–71. ISSN: 17919320. DOI: 10.25103/jestr.051.12. URL: https://www.researchgate.net/profile/Antanas_Cenys/publication/298598058_Overview_of_Real-Time_Antivirus_Scanning_Engines/links/56f2915208aee034d8c63a6a/Overview-of-Real-Time-Antivirus-Scanning-Engines.pdf.

[8] M. I. Al-Saleh and J. R. Crandall. "Application-level reconnaissance: timing channel attacks against antivirus software". In: *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats* (2011), pp. 9–9. URL: http://static.usenix.org/legacy/events/leet11/tech/full_papers/Al-Saleh.pdf.

[9] J. Eronen, K. Karjalainen, R. Puuperä, E. Kuusela, K. Halunen, M. Laakso, and J. Röning. "Software vulnerability vs. critical infrastructure-a case study of antivirus software". In: *International Journal on Advances in Security* 2.1 (2009), pp. 72–89.

[10] F. Xue. "Attacking Antivirus". In: *Black Hat* (2008). ISSN: 07400551. DOI: 10.1109/DMAMH.2007.33. URL: http://blackhat.com/presentations/bh-europe-08/Feng-Xue/Presentation/bh-eu-08-xue.pdf.

[11] Computer Profile. *McAfee remains the market leader in business antivirus solutions in Belgium, but is losing market share*. URL: https://www.computerprofile.com/analytics-papers/mcafee-remains-the-market-leader-in-business-antivirus-solutions-in-belgium-but-is-losing-market-share/ (visited on 01/17/2019).

[12] Das Malwerk. *Your one stop shop for fresh malware samples*. URL: http://dasmalwerk.eu/ (visited on 01/28/2019).

[13] A. Cortesi, M. Hils, and T. Kriechbaumer. *mitmproxy*. URL: https://mitmproxy.org/ (visited on 01/09/2019).

[14] R. Oppliger, R. Hauser, and D. Basin. "SSL/TLS session-aware user authentication - Or how to effectively thwart the man-in-the-middle". In: *Computer Communications* 29.12 (2006), pp. 2238–2246. ISSN: 01403664. DOI: 10.1016/j.comcom.2006.03.004.

[15] National Cyber Security Centre (NCSC). "TLS interception". In: 5 (2017). URL: https://www.ncsc.nl/english/current-topics/factsheets/factsheet-tls-interception.html.

[16] C. Evans and C.Palmer. *Certificate Pinning Extension for HSTS draft-evans-palmer-hsts-pinning-00*. URL: https://tools.ietf.org/html/draft-evans-palmer-hsts-pinning-00 (visited on 01/15/2019).

[17] Information Processing Fraunhofer Institute for Communication and Ergonomics (FKIE). "White paper encrypted traffic management". In: (2016), p. 33. URL: https://www.symantec.com/content/dam/symantec/docs/white-papers/fraunhofer-report-encrypted-traffic-management.pdf.

[18] K. Townsend. "Does it Matter if It's Black or White?" In: *Infosecurity* 8.3 (2011), pp. 36–39. ISSN: 1754-4548. DOI: https://doi.org/10.1016/S1754-4548(11)70039-0. URL: http://www.sciencedirect.com/science/article/pii/S1754454811700390.

[19] S. Mansfield-Devine. "The promise of whitelisting". In: *Network Security* 2009.7 (2009), pp. 4–6. ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(09)70085-6. URL: http://www.sciencedirect.com/science/article/pii/S1353485809700856.

[20] Aviad Cohen and Nir Nissim. "Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory". In: *Expert Systems with Applications* 102 (2018), pp. 158–178. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.02.039. URL: http://www.sciencedirect.com/science/article/pii/S0957417418301283.

[21] Jose Andre Morales, Michael Main, Weiliang Luo, Shouhuai Xu, and Ravi Sandhu. "Building malware infection trees". In: *Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software, Malware 2011* 9 (2011), pp. 50–57. DOI: 10.1109/MALWARE.2011.6112326.

[22] Sudhir Kumar Pandey and B. M. Mehtre. "Performance of malware detection tools: A comparison". In: *Proceedings of 2014 IEEE International Conference on Advanced Communication, Control and Computing Technologies, ICACCCT 2014* 978 (2015), pp. 1811–1817. DOI: 10.1109/ICACCCT.2014.7019422.

[23] Symantec. *Required exclusions for proxy servers to allow Endpoint Protection to connect to reputation and licensing servers*. URL: https://support.symantec.com/en_US/article.TECH162286.html (visited on 01/19/2019).

[24] Y. Nativ, L. Ludar, and 5fingers. *A repository of LIVE malwares for your own joy and pleasure*. 2015. URL: https://github.com/ytisf/theZoo (visited on 01/28/2019).

[25] faculty of Computer Science The University of Arizona. *lynx: Analysis of Hard-to-analyze Code*. URL: http://www2.cs.arizona.edu/projects/lynx-project/ (visited on 01/28/2019).

[26] EICAR. *The Anti-Malware Testfile*. URL: https://www.eicar.org/?page_id=3950 (visited on 01/28/2019).

APPENDIX A
AV TRAFFIC GENERATION MALWARE SAMPLES

TABLE 1
Das Malwerk malware samples

Number	Malware samples
1	1e84ff45-414b-11e8-b837-80e65024849a.file
2	1f7b55c7-414b-11e8-b18b-80e65024849a.file
3	230a6f87-414b-11e8-a52a-80e65024849a.file
4	266a11f5-414b-11e8-9ac8-80e65024849a.file
5	25786c51-414b-11e8-a472-80e65024849a.file

TABLE 2
Deloitte malware samples

Number	Malware samples
1	beacon-x64-test.exe
2	beacon-x86-test.dll
3	WindowsHost.dll
4	default-beacon.exe
5	default-beacon.dll
6	msf-vnm.exe

APPENDIX B
MALWARE SAMPLES STATIC AND DYNAMIC ANALYSIS

The first ten samples are obtained from Das Malwerk [12] and are True Positives (TPs). The samples 11 through 19 are obtained from the Zoo [24] and also are TPs. The samples 20 through 34 are from the University of Arizona [25] and the last four samples are of EICAR [26]. The samples of the University of Arizona and EICAR are False Positives (FPs). In table 3, S stand for Static and D stands for Dynamic.

TABLE 3
Static and Dynamic analysis of the AV vendors

Number	Malware samples	harmful	Symantec S	Symantec D	Trend Micro S	Trend Micro D	McAfee S	McAfee D
1	1c9877e3-414b-11e8-9653	TP	v	x	v	x	x	x
2	1e84ff45-414b-11e8-b837	TP	v	v	v	v	v	x
3	1f7b55c7-414b-11e8-b18b	TP	v	v	v	v	v	v
4	20db5785-414b-11e8-b3a7	TP	v	v	v	v	x	x
5	21f5d45c-414b-11e8-bfe9	TP	v	v	v	v	x	x
6	230a6f87-414b-11e8-a52a	TP	v	v	v	x	x	x
7	266a11f5-414b-11e8-9ac8	TP	v	v	v	v	x	x
8	2473c2ca-414b-11e8-8f4a	TP	v	v	v	v	x	x
9	2830f25c-414b-11e8-8ff2	TP	v	v	v	v	x	x
10	25786c51-414b-11e8-a472	TP	v	v	v	v	x	x
11	Artemis	TP	v	v	v	v	v	x
12	Win32.AgentTesla	TP	v	v	v	v	v	v
13	Win32.Unknown_SpectreMeltdown	TP	v	x	v	x	v	x
14	Win32.Unnamed_SpecMelt	TP	v	x	x	x	v	x
15	Win32.Vobfus	TP	v	v	v	v	v	x
16	Win32.WannaPeace	TP	v	v	v	v	v	x
17	Win32.Zurgop	TP	v	v	x	v	v	x
18	Win32Dircrypt.Trojan.Ransom.ABZ	TP	v	v	v	v	v	v
19	Win64.Trojan.GreenBug	TP	v	x	v	v	x	x
20	binary_search.exe	FP	x	x	x	x	x	x
21	blaster.exe	FP	v	x	v	x	v	x
22	bubble_sort.exe	FP	x	x	x	x	x	x
23	cairuh.exe	FP	v	x	x	x	x	x
24	epo.exe	FP	x	x	x	x	x	x
25	huffman.exe	FP	x	x	x	x	x	x
26	hunatcha.exe	FP	x	x	x	x	x	x
27	matrix_multiple.exe	FP	x	x	x	x	x	x
28	netsky1.exe	FP	v	x	v	x	x	x
29	netsky2.exe	FP	v	x	v	x	x	x
30	netsky3.exe	FP	x	x	x	x	x	x
31	netsky4.exe	FP	x	x	x	x	x	x
32	newstar.exe	FP	x	x	x	x	x	x
33	newstar-infect.exe	FP	v	x	x	x	x	x
34	tinyRISC-binary_search.exe	FP	x	x	x	x	x	x
35	eicar.com	FP	v	x	v	x	v	x
36	eicar.com.txt	FP	v	x	v	x	v	x
37	eicar_com.zip	FP	v	x	v	x	v	x
38	eicar_com2.zip	FP	v	x	v	x	v	x

APPENDIX C
OBSERVED ENDPOINTS
C.1 McAfee

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
a-ring-fallback.msedge.net	2	GET	0	43	0.03
ajax.googleapis.com	1	GET	0	93868	0.01
analytics.ccs.mcafee.com	7	POST	47232	0	0.1
analyticsdcs.ccs.mcafee.com	13	POST	38083	0	0.18
api-s2s.taboola.com	2	GET	0	0	0.03
app.mcafee.com	101	GET, POST	63	21340	1.37
arc.msn.com	4	GET, POST	1078	3172	0.05
b-ring.msedge.net	4	GET	0	43	0.05
bgpdefault-amb.msedge.net	2	GET	0	43	0.03
bing.com	1	GET	0	214	0.01
blob.weather.microsoft.com	1	GET	0	4265	0.01
c.bing.com	1	GET	0	0	0.01
c.msn.com	3	GET	0	28	0.04
cdn.adsoptimal.com	10	GET	0	89	0.14
cdn.onenote.net	2	GET	0	15	0.03
chart.googleapis.com	1	GET	0	2688	0.01
checkappexec.microsoft.com	1	POST	1348	143	0.01
cohort.ccs.mcafee.com	22	POST	712	0	0.3
consumerapps.mcafee.com	6	POST	867	2552	0.08
context-enroll.ccs.mcafee.com	1	POST	7206	0	0.01
ctldl.windowsupdate.com	10	GET	0	7091	0.14
cu1pehnswad01.servicebus.windows.net	5	POST	6409	0	0.07
cu1pehnswss01.servicebus.windows.net	364	POST	2220	0	4.94
dasmalwerk.eu	8	GET	0	73427	0.11
data.hackerwatch.org	6	GET	0	55	0.08
download.mcafee.com	276	GET, HEAD	0	125018	3.75
europe.smartscreen-prod.microsoft.com	2	GET, POST	674	4314	0.03
fls-na.amazon-adsystem.com	5	GET	0	43	0.07
fp.msedge.net	4	GET	0	0	0.05
fs.microsoft.com	2	GET, HEAD	0	28	0.03
g.live.com	4	GET, HEAD	0	0	0.05
google-analytics.com	373	POST	2080	35	5.07
home.mcafee.com	14	GET, POST	2638	3001	0.19
iecvlist.microsoft.com	2	GET	0	72323	0.03
images.outbrainimg.com	5	GET	0	16520	0.07
images.taboola.com	5	GET	0	39199	0.07
img-prod-cms-rt-microsoft-com.akamaized.net	2	GET	0	360215	0.03
img-s-msn-com.akamaized.net	161	GET	0	7047	2.19
ir-na.amazon-adsystem.com	5	GET	0	42	0.07
lam.rest.gti.mcafee.com	6	POST	516	76	0.08
login.live.com	8	POST	4964	10494	0.11
mcdp-chidc2.outbrain.com	4	GET	0	4	0.05
mcloud.mcafee.com	22	POST	6530	562	0.3
md5.hackerwatch.org	2	GET	0	476	0.03
messages.mcafee.com	11	GET, PATCH	210	46676	0.15
messaging.ccs.mcafee.com	24	GET	0	55	0.33
ocsp.digicert.com	5	GET	0	471	0.07
oneclient.sfx.ms	4	GET, HEAD	0	200	0.05
otf.msn.com	25	GET, OPTIONS, POST	774	8	0.34
peer2-amb.msedge.net	2	GET	0	43	0.03

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
platform.mcafee.com	11	GET, POST	261	244	0.15
policy.ccs.mcafee.com	17	POST	214	619	0.23
policyorchestration.ccs.mcafee.com	8	POST	965	2659	0.11
provision.ccs.mcafee.com	4	GET	0	1457	0.05
remote.vroptimal-3dx-assets.com	10	GET	0	0	0.14
resources.infolinks.com	2	GET	0	398194	0.03
ris.api.iris.microsoft.com	5	GET	0	0	0.07
router.infolinks.com	10	GET	0	0	0.14
s-ring.msedge.net	2	GET	0	43	0.03
sadownload.mcafee.com	136	GET	0	61567	1.85
sam.msn.com	4	OPTIONS, POST	427	0	0.05
servicediscovery.ccs.mcafee.com	17	POST	352	739	0.23
sm.mcafee.com	1	GET	0	3955	0.01
ssl.google-analytics.com	6	GET	0	7742	0.08
static-spartan-neu-s-msn-com.akamaized.net	17	GET	0	69565	0.23
store-images.s-microsoft.com	1	GET	0	1841	0.01
storeedgefd.dsx.mp.microsoft.com	1	GET	0	126	0.01
tags.tiqcdn.com	3	GET	0	102771	0.04
tile-service.weather.microsoft.com	3	GET	0	4231	0.04
us.mcafee.com	10	POST	3621	1509	0.14
virustotalcloud.appspot.com	47	GET	0	9256	0.64
webadvisorc.rest.gti.mcafee.com	5	POST	228	66	0.07
ws-na.amazon-adsystem.com	5	GET	0	15208	0.07
ws.mcafee.com	2	POST	800	385	0.03
wss.rest.gti.mcafee.com	5382	POST	511	105	73.11
www.bing.com	59	GET, POST	3144	9705	0.8
www.mcafee.com	4	GET	0	58937	0.05
www.msn.com	19	GET, POST	504	61614	0.26
www.virustotal.com	20	GET	0	21433	0.27

C.2 Symmantec

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
a-ring-fallback.msedge.net	2	GET	0	43	0.28
a-ring.msedge.net	4	GET	0	43	0.56
access.backup.norton.com	8	GET	0	223	1.11
ajax.googleapis.com	1	GET	0	97163	0.14
api-s2s.taboola.com	1	GET	0	0	0.14
b-ring.msedge.net	2	GET	0	43	0.28
c.s-microsoft.com	1	GET	0	3560	0.14
cdn.adsoptimal.com	2	GET	0	89	0.28
cdn.tt.omtrdc.net	1	GET	0	43582	0.14
checkappexec.microsoft.com	1	POST	974	271	0.14
cm.everesttech.net	1	GET	0	0	0.14
ctldl.windowsupdate.com	4	GET	0	3592	0.56
dasmalwerk.eu	18	GET	0	99954	2.51
dls.symantec.com	1	GET	0	0	0.14
dpm.demdex.net	3	GET	0	136	0.42
europe.smartscreen-prod.microsoft.com	1	GET	0	8358	0.14
faults.norton.com	11	POST	1391	0	1.53
fls-na.amazon-adsystem.com	1	GET	0	43	0.14
fonts.googleapis.com	2	GET	0	0	0.28
fonts.gstatic.com	4	GET	0	21873	0.56

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
fp.msedge.net	4	GET	0	0	0.56
hb.lifecycle.norton.com	3	GET, POST	728	39	0.42
identitysafe.norton.com	12	GET	0	104555	1.67
iecvlist.microsoft.com	2	GET	0	72323	0.28
images-na.ssl-images-amazon.com	1	GET	0	2679	0.14
images.taboola.com	7	GET	0	52796	0.97
img-prod-cms-rt-microsoft-com.akamaized.net	5	GET	0	10804	0.7
img-s-msn-com.akamaized.net	144	GET	0	7919	20.06
ir-na.amazon-adsystem.com	1	GET	0	42	0.14
l-ring.msedge.net	2	GET	0	43	0.28
liveupdate.symantec.com	1	GET	0	1	0.14
liveupdate.symanteclive update.com	79	GET	0	12934	11.0
login.live.com	1	GET	0	0	0.14
login.microsoftonline.com	1	GET	0	961	0.14
markets.books.microsoft.com	1	GET	0	331	0.14
nexus.ensighten.com	4	GET	0	47549	0.56
ocsp.digicert.com	5	GET	0	471	0.7
ocsp.thawte.com	1	GET	0	1336	0.14
ocsp.verisign.com	1	GET	0	1754	0.14
oms.symantec.com	2	GET	0	46	0.28
otf.msn.com	49	GET, OPTIONS, POST	831	5	6.82
ow1.res.office365.com	2	GET	0	43	0.28
ratings-wrs.symantec.com	1	GET	0	210	0.14
remote.vroptimal-3dx-assets.com	2	GET	0	0	0.28
resources.infolinks.com	2	GET	0	399990	0.28
router.infolinks.com	2	GET	0	0	0.28
sam.msn.com	6	OPTIONS, POST	368	0	0.84
sf.symcd.com	1	GET	0	1660	0.14
shasta-rrs.symantec.com	23	POST	6065	2127	3.2
sitedirector.symantec.com	1	GET	0	715	0.14
spoc.norton.com	10	POST	118	5	1.39
ssl.google-analytics.com	4	GET	0	11595	0.56
static-spartan-neu-s-msn-com.akamaized.net	20	GET	0	58136	2.79
static.nortoncdn.com	2	GET	0	32354	0.28
stats.norton.com	82	GET, POST	0	13	11.42
storage.backup.norton.com	2	POST	2982	367	0.28
symantec.demdex.net	1	GET	0	6939	0.14
symantec.tt.omtrdc.net	1	GET	0	0	0.14
ts-ocsp.ws.symantec.com	1	GET	0	1469	0.14
updatecenter.norton.com	2	GET, POST	56	2758	0.28
wms-na.amazon-adsystem.com	3	GET	0	1708	0.42
ws-na.amazon-adsystem.com	1	GET	0	15208	0.14
www.bing.com	123	GET, POST	4782	26237	17.13
www.msn.com	30	GET, POST	622	49111	4.18
www2.bing.com	1	GET	0	64	0.14

C.3 Trendmicro

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
364bf5fa.akstat.io	2	GET, POST	4077	0	0.28
ajax.googleapis.com	1	GET	0	97163	0.14

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
arc.msn.com	8	GET, POST	181	652	1.11
blob.weather.microsoft.com	1	GET	0	8245	0.14
c.go-mpulse.net	1	GET	0	1771	0.14
carcharodon.trendmicro.com	2	POST	7746	62	0.28
cdn.adsoptimal.com	4	GET	0	89	0.56
cdn.onenote.net	1	GET	0	15	0.14
chk.trendmicro.com	1	GET	0	369	0.14
dasmalwerk.eu	20	GET	0	114411	2.79
displaycatalog.mp.microsoft.com	11	GET	0	30301	1.53
fls-na.amazon-adsystem.com	2	GET	0	43	0.28
fonts.googleapis.com	2	GET	0	0	0.28
fonts.gstatic.com	4	GET	0	21873	0.56
iecvlist.microsoft.com	3	GET	0	73205	0.42
images-na.ssl-images-amazon.com	1	GET	0	2679	0.14
images.outbrainimg.com	4	GET	0	13896	0.56
images.taboola.com	5	GET	0	37272	0.7
img-s-msn-com.akamaized.net	87	GET	0	9410	12.12
ipv6-mirror-iaus.activeupdate.trendmicro.com:443	47	GET	0	201123	6.55
ir-na.amazon-adsystem.com	2	GET	0	42	0.28
licensing.mp.microsoft.com	13	POST	758	526	1.81
login.live.com	12	POST	4975	10417	1.67
otf.msn.com	22	GET, OPTIONS, POST	809	11	3.06
pt31-feature-cfg-prod.s3.amazonaws.com	1	GET	0	134	0.14
purchase.mp.microsoft.com	6	POST, PUT	294	3137	0.84
remote.vroptimal-3dx-assets.com	4	GET	0	0	0.56
res.appletuner.trendmicro.com	1	GET	0	373	0.14
resources.infolinks.com	2	GET	0	399990	0.28
ris.api.iris.microsoft.com	8	GET	0	0	1.11
router.infolinks.com	4	GET	0	0	0.56
sam.msn.com	6	OPTIONS, POST	360	0	0.84
static-spartan-neu-s-msn-com.akamaized.net	16	GET	0	53202	2.23
storesdk.dsx.mp.microsoft.com	5	GET	0	284	0.7
tile-service.weather.microsoft.com	1	GET	0	4233	0.14
titanium15-0-en.url.trendmicro.com:80	6	GET	0	204	0.84
titanium15-en.gfrbridge.trendmicro.com:80	33	GET	0	99	4.6
titanium1500-en-census.trendmicro.com:80	35	GET	0	579	4.87
tms15.icrc.trendmicro.com	267	GET	0	3700	37.19
wms-na.amazon-adsystem.com	3	GET	0	1708	0.42
ws-na.amazon-adsystem.com	2	GET	0	15208	0.28
www.bing.com	51	GET, POST	1082	13060	7.1
www.msn.com	11	GET, POST	285	82922	1.53

C.4 Kaspersky

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
arc.msn.com	7	GET, POST	414	143	3.03
cdn.onenote.net	3	GET	0	15	1.3
checkappexec.microsoft.com	1	POST	1348	143	0.43
click.kaspersky.com	1	GET	0	0	0.43
displaycatalog.mp.microsoft.com	2	GET	0	28164	0.87
dnl-10.geo.kaspersky.com	1	GET	0	25	0.43

Domain name	Number of requests	Observed methods	Average request length (bytes)	Average response length (bytes)	Percentage of total number of requests
dnl-12.geo.kaspersky.com	36	GET	0	4277	15.58
dnl-13.geo.kaspersky.com	3	GET	0	730	1.3
europe.smartscreen-prod.microsoft.com	1	POST	1348	143	0.43
images.taboola.com	2	GET	0	266107	0.87
img-s-msn-com.akamaized.net	38	GET	0	9106	16.45
img.s-msn.com	1	GET	0	410	0.43
licensing.mp.microsoft.com	10	POST	761	293	4.33
login.live.com	1	POST	4915	10749	0.43
otf.msn.com	11	GET, OPTIONS, POST	795	8	4.76
ris.api.iris.microsoft.com	9	GET	0	0	3.9
sam.msn.com	8	OPTIONS, POST	355	0	3.46
sls.update.microsoft.com	30	GET	0	7793	12.99
static-spartan-neu-s-msn-com.akamaized.net	8	GET	0	47433	3.46
support.kaspersky.co.uk	39	GET	0	15039	16.88
tile-service.weather.microsoft.com	3	GET	0	4244	1.3
www.bing.com	8	GET, POST	590	4729	3.46
www.msn.com	8	GET, POST	604	87628	3.46

C.5 Common

Domain name	Different datasets observed in	Total number of requests
364bf5fa.akstat.io	1	2
a-ring-fallback.msedge.net	2	4
a-ring.msedge.net	1	4
access.backup.norton.com	1	8
ajax.googleapis.com	3	3
analytics.ccs.mcafee.com	1	7
analyticsdcs.ccs.mcafee.com	1	13
api-s2s.taboola.com	2	3
app.mcafee.com	1	101
arc.msn.com	3	19
b-ring.msedge.net	2	6
bgpdefault-amb.msedge.net	1	2
bing.com	1	1
blob.weather.microsoft.com	2	2
c.bing.com	1	1
c.go-mpulse.net	1	1
c.msn.com	1	3
c.s-microsoft.com	1	1
carcharodon.trendmicro.com	1	2
cdn.adsoptimal.com	3	16
cdn.onenote.net	3	6
cdn.tt.omtrdc.net	1	1
chart.googleapis.com	1	1
checkappexec.microsoft.com	3	3
chk.trendmicro.com	1	1
click.kaspersky.com	1	1
cm.everesttech.net	1	1
cohort.ccs.mcafee.com	1	22
consumerapps.mcafee.com	1	6
context-enroll.ccs.mcafee.com	1	1
ctldl.windowsupdate.com	2	14
cu1pehnswad01.servicebus.windows.net	1	5
cu1pehnswss01.servicebus.windows.net	1	364

Domain name	Different datasets observed in	Total number of requests
dasmalwerk.eu	3	46
data.hackerwatch.org	1	6
displaycatalog.mp.microsoft.com	2	13
dls.symantec.com	1	1
dnl-10.geo.kaspersky.com	1	1
dnl-12.geo.kaspersky.com	1	36
dnl-13.geo.kaspersky.com	1	3
download.mcafee.com	1	276
dpm.demdex.net	1	3
europe.smartscreen-prod.microsoft.com	3	4
faults.norton.com	1	11
fls-na.amazon-adsystem.com	3	8
fonts.googleapis.com	2	4
fonts.gstatic.com	2	8
fp.msedge.net	2	8
fs.microsoft.com	1	2
g.live.com	1	4
google-analytics.com	1	373
hb.lifecycle.norton.com	1	3
home.mcafee.com	1	14
identitysafe.norton.com	1	12
iecvlist.microsoft.com	3	7
images-na.ssl-images-amazon.com	2	2
images.outbrainimg.com	2	9
images.taboola.com	4	19
img-prod-cms-rt-microsoft-com.akamaized.net	2	7
img-s-msn-com.akamaized.net	4	430
img.s-msn.com	1	1
ipv6-mirror-iaus.activeupdate.trendmicro.com:443	1	47
ir-na.amazon-adsystem.com	3	8
l-ring.msedge.net	1	2
lam.rest.gti.mcafee.com	1	6
licensing.mp.microsoft.com	2	23
liveupdate.symantec.com	1	1
liveupdate.symantecliveupdate.com	1	79
login.live.com	4	22
login.microsoftonline.com	1	1
markets.books.microsoft.com	1	1
mcdp-chidc2.outbrain.com	1	4
mcloud.mcafee.com	1	22
md5.hackerwatch.org	1	2
messages.mcafee.com	1	11
messaging.ccs.mcafee.com	1	24
nexus.ensighten.com	1	4
ocsp.digicert.com	2	10
ocsp.thawte.com	1	1
ocsp.verisign.com	1	1
oms.symantec.com	1	2
oneclient.sfx.ms	1	4
otf.msn.com	4	107
ow1.res.office365.com	1	2
peer2-amb.msedge.net	1	2
platform.mcafee.com	1	11
policy.ccs.mcafee.com	1	17
policyorchestration.ccs.mcafee.com	1	8
provision.ccs.mcafee.com	1	4
pt31-feature-cfg-prod.s3.amazonaws.com	1	1
purchase.mp.microsoft.com	1	6
ratings-wrs.symantec.com	1	1
remote.vroptimal-3dx-assets.com	3	16

Domain name	Different datasets observed in	Total number of requests
res.appletuner.trendmicro.com	1	1
resources.infolinks.com	3	6
ris.api.iris.microsoft.com	3	22
router.infolinks.com	3	16
s-ring.msedge.net	1	2
sadownload.mcafee.com	1	136
sam.msn.com	4	24
servicediscovery.ccs.mcafee.com	1	17
sf.symcd.com	1	1
shasta-rrs.symantec.com	1	23
sitedirector.symantec.com	1	1
sls.update.microsoft.com	1	30
sm.mcafee.com	1	1
spoc.norton.com	1	10
ssl.google-analytics.com	2	10
static-spartan-neu-s-msn-com.akamaized.net	4	61
static.nortoncdn.com	1	2
stats.norton.com	1	82
storage.backup.norton.com	1	2
store-images.s-microsoft.com	1	1
storeedgefd.dsx.mp.microsoft.com	1	1
storesdk.dsx.mp.microsoft.com	1	5
support.kaspersky.co.uk	1	39
symantec.demdex.net	1	1
symantec.tt.omtrdc.net	1	1
tags.tiqcdn.com	1	3
tile-service.weather.microsoft.com	3	7
titanium15-0-en.url.trendmicro.com:80	1	6
titanium15-en.gfrbridge.trendmicro.com:80	1	33
titanium1500-en-census.trendmicro.com:80	1	35
tms15.icrc.trendmicro.com	1	267
ts-ocsp.ws.symantec.com	1	1
updatecenter.norton.com	1	2
us.mcafee.com	1	10
virustotalcloud.appspot.com	1	47
webadvisorc.rest.gti.mcafee.com	1	5
wms-na.amazon-adsystem.com	2	6
ws-na.amazon-adsystem.com	3	8
ws.mcafee.com	1	2
wss.rest.gti.mcafee.com	1	5382
www.bing.com	4	241
www.mcafee.com	1	4
www.msn.com	4	68
www.virustotal.com	1	20
www2.bing.com	1	1

APPENDIX D
USER EMULATION SCRIPTS

D.1 pywinauto script

```
import pywinauto, time

app = pywinauto.Application(backend='uia').connect(path='explorer')
app.Program_manager.desktop.Microsoft_edge.click_input(double=True)

time.sleep(1)
edge_app = pywinauto.Application(backend='uia').connect(title_re='.* Microsoft Edge')

edge_app.Microsoft_edge.child_window(auto_id='addressEditBox').set_edit_text('http://192.168.3.1:8000').type_keys('ENTER')
time.sleep(3)
edge_app.top_window().descendants(title='beacon-x64-test.exe', control_type='Hyperlink')[0].click_input()
edge_app.top_window().descendants(title='Save', control_type='Button')[0].click_input()
time.sleep(2)
edge_app.kill()

app.Taskbar.Running_applications.File_explorer.click_input()
time.sleep(1)

file_explorer = pywinauto.Application(backend='uia').connect(title='File Explorer')
file_explorer.File_explorer.File_explorer.Tree_view.Desktop.This_Pc.Downloads.click_input()
file_explorer.Downloads.descendants(title='beacon-x64-test.exe', control_type='ListItem')[0].click_input(double=True)
time.sleep(1)
if file_explorer.Downloads.Open_File_Security_Warning.exists():
file_explorer.Downloads.Open_File_Security_Warning.Run.set_focus()
time.sleep(1)
file_explorer.Downloads.Open_File_Security_Warning.Run.set_focus().click_input()
```

D.2 pyautogui script

```
import webbrowser, time, pyautogui as p

p.FAILSAFE = True
p.PAUSE = 1

class iexplorer(object):
def open_iexplorer(self):
webbrowser.open('http://192.168.3.1:8000')
time.sleep(1)

def Select_sample(self, x, y):
p.moveTo(x,y, duration=1)
p.click(clicks=1, button='left')

def download_malware(self):
save = p.locateOnScreen(r'C:/Users/RP1/Pictures/Save.png')
if save:
coordinates=[]
for x in save:
coordinates.append(x)
p.moveTo(coordinates[0], coordinates[1], duration=0.5)
p.click(clicks=1, button='left')
```

```
def mcafee_popup(self):
time.sleep(2)
mcafee = p.locateOnScreen(r'C:/Users/RP1/Pictures/accept_the_risk.png')
if mcafee:
coordinates=[]
for x in mcafee:
coordinates.append(x)
p.moveTo(coordinates[0], coordinates[1], duration=0.5)
p.click(clicks=1, button='left')

def open_folder(self):
time.sleep(15)
folder = p.locateOnScreen(r'C:/Users/RP1/Pictures/Open_folder.png')
if folder:
coordinates=[]
for x in folder:
coordinates.append(x)
p.moveTo(coordinates[0], coordinates[1], duration=0.5)
p.click(clicks=1, button='left')

def run_file(self):
time.sleep(2)
exe = p.locateOnScreen(r'C:/Users/RP1/Pictures/exe.png')
if exe:
coordinates=[]
for x in exe:
coordinates.append(x)
p.moveTo(coordinates[0], coordinates[1], duration=0.5)
p.click(clicks=2, button='left')
time.sleep(1)

smartscreen = p.locateOnScreen(r'C:/Users/RP1/Pictures/smartscreen.png')
if smartscreen:
coordinates=[]
for x in smartscreen:
coordinates.append(x)
p.moveTo(coordinates[0], coordinates[1], duration=0.5)
p.click(clicks=1, button='left')

browser = iexplorer()
browser.open_iexplorer()
browser.select_sample_sample(150,225)
browser.download_malware()
browser.open_folder()
browser.run_file()
```