TRUMANBOX: Improving Dynamic Malware Analysis by Emulating the Internet

Christian Gorecki[1⋆], Felix C. Freiling[2], Marc Kührer[3], and Thorsten Holz[3]

[1] AGT Germany
[2] Friedrich-Alexander-University Erlangen-Nuremberg
[3] Ruhr-University Bochum

Abstract. Dynamic analysis of malicious software (*malware*) is a powerful tool in countering modern threats on the Internet. In dynamic analysis, a malware sample is executed in a controlled environment and its actions are logged. Through dynamic analysis, an analyst can quickly obtain an overview of malware behavior and can decide whether or not to indulge into tedious manual analysis of the sample. However, usual dynamic analysis exposes the Internet to the threats of an executed malware (like portscans) because advanced concealment techniques of malware often require full Internet access. For example, a missing link to the Internet or the unavailability of a specific server often causes the malware to not trigger its malicious behavior. In this paper, we present TRUMANBOX, a technique to emulate relevant parts of the Internet to enhance dynamic malware analysis. We show that TRUMANBOX not only prevents many threats but also enlarges the scope of the types of malware that can be analyzed dynamically.

1 Introduction

Since the amount of malicious programs is increasing day by day, many different approaches have been developed to analyze malware. Until recently, the standard approach to malware analysis was based on *static* methods, i.e., disassembling the binary and manual reverse engineering. Modern malware increasingly thwarts such analysis through refined encryption and obfuscation methods. Therefore, *dynamic analysis* methods offer a promising approach to analyze malicious software.

In dynamic analysis, the actual execution of a malware sample is an important part of the analysis process. However, executing malware exposes the runtime environment to malicious activity. Especially malicious outgoing network traffic induced by the corresponding malware samples in execution is a critical issue. Simply preventing any access to the Internet, however, is also not feasible in many cases since malware often needs Internet access to trigger and exhibit its malicious behavior (that is to be analyzed). For example, the unavailability of a specific command and control server often causes certain bots to remain silent.

1.1 Related Work

The issues of preventing outgoing malicious traffic from an analysis platform are similar to those experienced with *honeypots* [10, 12]. Briefly spoken, a honeypot is electronic

⋆ Work by Christian Gorecki was performed while being with University of Mannheim.

bait that can be used for malware capture, i.e., a resource on the Internet whose use lies in the fact that it is compromised. Especially *high-interaction* honeypots, i.e., those that emulate full systems, are a threat to an analysis network. To protect the environment from malicious traffic, a honeynet is separated from the Internet by a *Honeywall* [3]. A Honeywall typically is an OSI layer-2 bridging device with different capabilities in logging and filtering. Often, such a device is called an *extrusion prevention system*.

HONEYD is a simulation framework that allows to instrument thousands of IP addresses with virtual honeypots running corresponding network services [11]. Due to the simplicity of configuration, simulation of huge networks can be set up within minutes and huge networks can be simulated with high-performance. Even though HONEYD is very flexible in its configuration, it is static in the way that services are only listening on predefined IP addresses and on static ports that need to be specified in advance.

As mentioned above, we focus on extrusion prevention not during malware capture but rather during malware analysis. In particular, we are interested in dynamic malware analysis, i.e., running malware within a protected "sandbox" environment like CWSANDBOX [13] and ANUBIS [2, 7] which log system calls and thereby create traces of the visible behavior of that malware. As mentioned above, running malware makes it necessary to simulate parts of the network to stimulate certain behavior. One system that does this is the *Botnet Evaluation Environment* (BEE) [1], a testbed for evaluating bots and botnets in a self-contained environment based on *Emulab* [5]. Emulab is a platform for creating virtual network nodes, which can emulate operating systems or applications after being equipped with a corresponding image. Overall, BEE offers services such as IRC, DNS, and DHCP. These services are available on certain IP addresses connected to the virtual network, and outgoing traffic is redirected to these services. For example, any IRC request independent from the original destination address will contact the IRC server deployed within the network. As this network translation is implemented on the client, this approach requires control of the clients. We overcome this drawback in our work, as we will see later on.

Last we want to mention INetSim [6], a very powerful network simulation suite, particularly regarding the number of different services implemented. However, INetSim assumes that protocols always use their "correct" port (e.g., HTTP on port 80), which is not a valid assumption when analyzing malware.

1.2 Contributions

The challenge is to find a trade-off between taking the risk of infecting third party systems and a reduced behavior of malware by preventing or restricting outgoing traffic. In this paper, we present the design, implementation, and evaluation of TRUMANBOX, a system that provides novel flexibility in malware analysis.

The idea of TRUMANBOX is to emulate the most commonly used Internet services on one physical machine in an easily configurable way. Like Honeywall, the system is inserted into the Internet connection as a transparent network bridge. Like BEE, the system emulates network nodes to provide services like IRC. However, unlike Honeywall, BEE, and INetSim, the system (1) dispatches all supported service requests – independent from destination IP address and TCP port – to the corresponding local service, and (2) uses a heuristic pattern matching approach to identify protocols when they are used

on non-standard ports. This means that with TRUMANBOX, any service can be bound to every available port.

TRUMANBOX also offers a flexible set of working modes that range from a behavior analogous to Honeywall to a full Internet emulation that does not need real Internet connectivity at all. The overall goal of these functions is to keep malware running as long as possible, without actually contacting the Internet, or at least restricting/controlling outgoing traffic, to prevent malicious interactions with third-party systems. From the view of an analyst, especially the full emulation mode is a notable feature of TRUMANBOX, since it allows comprehensive malware analysis without Internet access.

To summarize, our contributions are twofold:

- We present the design and implementation of TRUMANBOX, a novel and flexible tool to support dynamic malware analysis. TRUMANBOX has been implemented under Linux and is freely available [4].
- We evaluate TRUMANBOX and show that it offers additional flexibility to malware analysis not offered by other existing tools. For example, in full emulation mode, we found that in 98 out of 154 test cases (malware samples) the traffic reports include at least the same information we would obtain by using a sandbox environment like ANUBIS and CWSANDBOX. Furthermore, 36 TRUMANBOX reports even included network traffic which was not logged by these sandbox environments at all. Note that these results were achieved *without* connection to the Internet.

1.3 Roadmap

This work is outlined as follows: In Section 2, we provide background on different techniques used in our implementation. In Section 3, we describe the actual system. We evaluate the system in Section 4 and conclude in Section 5.

2 Technical Background

We now give some technical background that is important for the understanding of TRUMANBOX in Section 3.

2.1 Naming Conventions

Since TRUMANBOX is a bridge with emulation capabilities, we use two network interface cards (NICs) in our machine, one to the client side we want to "provide" with the emulation and one to the outside network infrastructure, e.g., the Internet. We name those NICs `eth1` and `eth0`, respectively (see Fig. 1). The logical bridge device containing these interfaces as so-called bridge-ports is called `br0`. The *client* is usually a computer running the malware (possibly within a malware analysis sandbox), however, the network structure on the client side can vary.

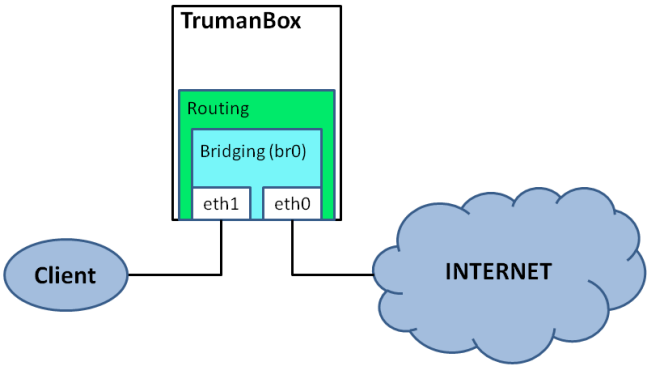

Fig. 1: Naming conventions

2.2 Bridging and (Re-)directing Data

While technically TRUMANBOX is a transparent bridge, it also has to behave as a router if traffic is redirected within the Internet emulation. A transparent bridge has usually no IP address of its own and thus cannot be detected by common port or network scanning tools, enabling a *stealthy* and *transparent* device for intercepting network data traffic.

Fig. 2 provides a simplified view about how a packet passes through the Linux kernel for the case of bridging the two available network interfaces. First, the incoming packets pass the *BROUTING* chain in the *broute* table, where, based on the destination MAC address, it will be decided to pass the packet to OSI layer-3 (IP protocol), to drop it, or to bridge it, i.e., which is an OSI layer-2 forwarding to the other interface. This decision depends on whether the destination MAC aims at connecting to our machine, a machine on the same side, or one that is known to be on the other side of the bridge, respectively. In case the location of the system with the corresponding MAC address is unknown, the packet is flooded over all forwarding bridge ports. Given this complex chain, there are multiple ways in which interception can be done, as we now explain.

We chose to intercept packets at the end of the *PREROUTING* chain in the *nat* table. The next hook, i.e., the *FORWARD* chain in the *filter* table of ebtables, is already part of the forwarding mechanism, hence it would be too late to intercept the packets using this or one of the following hooks. So we finally use appropriate iptables rules to create the possibility of data redirection.

However, using the iptables command, our bridge needs an IP address to which data can be redirected. Therefore, we have to configure an IP address to our logical interface `br0`. Since bridges usually do not have an IP address, we also have to take further steps for maintaining stealthiness.

Since the IP address of TRUMANBOX is needed only for internal packet forwarding, no other machine needs access to our system using the IP protocol. Therefore, we drop all incoming ARP broadcasts by using ebtables. In this way, our system stays invisible to the client side, while we can still access it remotely from the other interface pointing to the Internet. This is important, as we will need outgoing communication for certain modes of our implementation, as discussed later on.

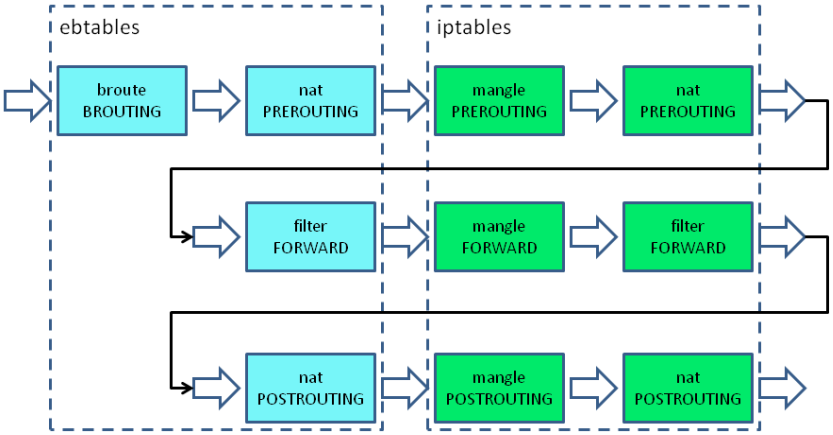

Fig. 2: Simplified data-packet traverse through the Linux kernel of a bridge.

2.3 Extracting Non-Altered Header Information

Combining the techniques introduced so far, we obtain a transparent bridge that can redirect bypassing traffic to itself. Unfortunately, we loose original TCP header information during this process because the destination IP address and (optionally) the TCP port are changed. So we had to employ a technical trick: We kept the *PREROUTING* chain in the *mangle* table (see Fig. 2) in which the non-altered header information is still visible. We then use the *QUEUE* target in iptables to hand data-packets to the userspace. To access the data (in particular the original header information of the packet), we use the library *libipq* [8] and then hand back the packet to the kernel. Furthermore, we store the information about original destination in a global variable and can use this information in the dispatching phase.

3 TRUMANBOX

3.1 Approach

Combining the techniques presented in Section 2, we can setup TRUMANBOX as a transparent (stealthy) bridge, redirecting selected (or all) bypassing traffic to itself. Thus, any outgoing malicious traffic can be avoided. In order to trick malware and have its malicious activities still being performed, we need to provide the impression of Internet connectivity. For this purpose, standard services like HTTP, FTP, IRC, and SMTP servers are provided (almost) in their standard configuration, e.g., standard listening port, etc., locally on TRUMANBOX.

To overcome static protocol identification driven by the TCP destination port, we provide more flexibility by a hybrid protocol identification. Certain malicious programs use non-standard ports for their communication, some malware samples even use standard protocols on a different standard port, e.g., HTTP on port 21 or FTP on port 80. To overcome this problem, we have a dispatching service running on a certain port

where all bypassing connections are redirected to. The dispatching service maps new connections to the local standard services, as described in Section 3.2, depending on the protocol in use. Information on the original destination, e.g., IP address, TCP port, etc., are extracted for further processing. The connection mapping/dispatching is done in respect to preserving the possibility of intercepting and manipulating payloads.

To improve the emulation provided by TRUMANBOX, we allow adjustment of the amount of outgoing traffic by providing two different modes (see Fig. 3):

1. *Full Emulation* requires no Internet access at all and provides the client with a rather static and simple emulation of the Internet.
2. *Half Proxy* requires access to the Internet; however, TRUMANBOX only uses this to gather additional information to improve the emulation, e.g., by fetching banner information from the originally targeted destination.

Independent of the actual mode in use, the client never interacts with the Internet directly, but only communicates with services provided locally on the TRUMANBOX. DNS is not affected by the mode. Instead, it can be configured separately in order to en-/disable proper domain name resolution to corresponding IP addresses, during the analysis process.

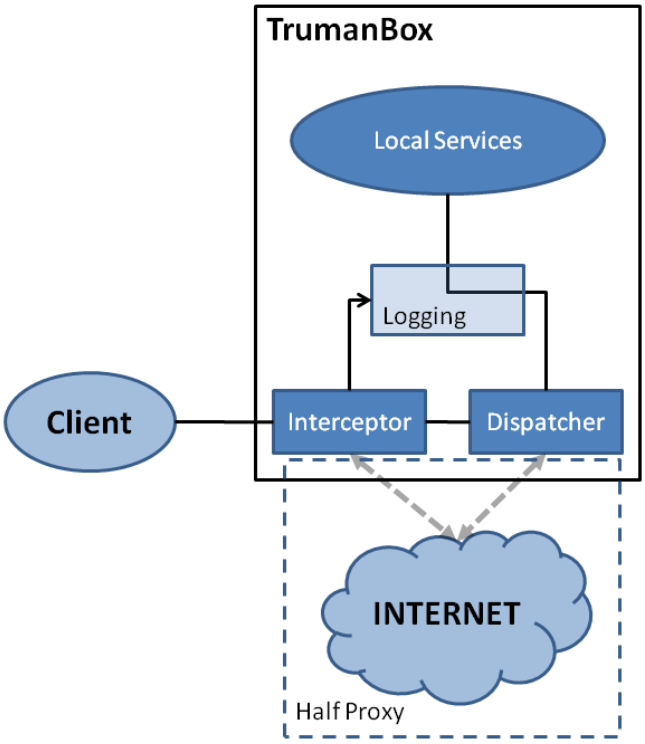

Fig. 3: Connection modes

Even though the *half proxy* mode appears to be an enhanced *full emulation*, there are reasons for having both modes implemented. In certain environments, execution of malware on a computer with full Internet access might be no option due to legal obligations. While execution without Internet access might result into no malicious actions being triggered, TRUMANBOX in *full emulation* may satisfy trigger conditions to render dynamic analysis being successful. Another scenario is dynamic analysis of, for instance, malicious software trying to reach for a command and control server that is not online anymore. Again, *full emulation* can improve dynamic analysis results. However, *half proxy* mode might be adequate when we are aware of the fact that investigated malware samples do not perform any malicious actions but downloading files and contacting the command and control servers.

More details on the applications of TRUMANBOX are available in Section 4.

3.2 Implementation Details

Header Extraction Even though our dispatching approach, which will be discussed in detail in the following, has some advantages, e.g., in terms of easily redirecting traffic, there is also one major drawback: it does neither preserve the original destination IP address nor the destination port. However, this information is important in postprocessing of dynamic malware analysis. Therefore, we have to extract the data as described in Section 2.3. Once, the information is extracted, it is used for logging and (in *half proxy* mode) for gathering further information from the original destination host, as for example service banners.

Dispatching The core of our implementation is the dispatching function. Its processing starts as soon as a new connection has been established to the local port it is listening on, e.g., TCP/UDP port 400. Right after accepting the new connection request (A), a second connection (B) is established depending on the protocol of the incoming connection and the mode the TRUMANBOX is currently running. In the following, payloads are forwarded between A and B. Depending on the mode of operation, certain alterations of payload and execution of additional functions are triggered by a successful match of predefined patterns against the payload. We will explain these different manipulations later when discussing the mode of operation they occur in.

Hybrid Protocol Identification In order to handle more than only one connection, we create a new process for every incoming connection. Since establishing the second connection (B) also depends on the protocol of the incoming connection (A), we somehow have to identify it. The common identification by considering the TCP destination port is not strong enough for our purposes because data connections initiated by malware often use non-standard ports. Therefore, we strengthen the traditional protocol identification by implementing a payload protocol identification which is applied first. These two algorithms in combination form our *hybrid protocol identification*: First we identify new connections by observing the payload; if this fails, we examine the destination port to determine the protocol in use.

To use the first payload of a connection for protocol identification, we first have to distinguish between server-first-sending (SFS) and client-first-sending (CFS) protocols. For example, in HTTP and IRC usually the client sends the first payload, hence these are CFS protocols. SMTP and FTP start with a payload sent by the server (often called the "banner" or "welcome message"), hence, these are examples for SFS protocols.

Given the first payload of a new connection, we use the patterns listed in Table 1 to determine the protocol. Here we do not distinguish between SFS and CFS protocols, since this would be an unnecessary restriction. Rather, we stay generic to also be capable of recognizing possible modifications of SFS protocols to corresponding CFS versions. If all these methods fail, we try to determine the protocol by considering the destination port even though, particularly in malware analysis, this is not reliable in all cases.

Table 1: Payload patterns determining the protocol

Protocol	Pattern (at the beginning)	Pattern (somewhere)
HTTP	`"GET /"`	
IRC	`"NICK "`	
FTP	`"220 "`	`"ftp"` (case insensitive)
SMTP	`"220"`	`"mail"` OR `"smtp"` (case insensitive)

Sometimes there is no payload coming over the incoming connection which makes protocol identification harder. We can alleviate this in half proxy mode of TRUMANBOX, as we explain below.

Full Emulation Mode The goal of full emulation mode is to emulate the Internet without any access to the Internet. This is probably the most challenging of our modes: we want to prevent any kind of data connection (apart from possibly DNS) to the Internet without letting the client (i.e., the malware) become aware of it. Therefore, we provide local services as generic as possible and redirect the connection requests of the client to our machine. If outgoing DNS traffic has to be avoided as well, we are forced to also set up a local DNS service which resolves requested hostnames to predefined IP addresses.

Since responses of our generic services are static, it is quite easy to fingerprint TRUMANBOX, so this mode is not meant to trick a human intruder or some malware which uses methods to verify the authentication of the server to contact. For instance, a rather simple method of detecting TRUMANBOX is to send requests to unreachable servers. In full emulation mode, TRUMANBOX always responds with a valid reply as if the originally targeted server is online. Since the attacker does not expect a response at all, he might realize that some interception is going on. A further approach could be to investigate the content of received responses. Whenever an attacker requests a specific file from a webserver, e.g. an update of a malware executable, TRUMANBOX returns fake content since in full emulation mode, it is not allowed to contact the Internet to get the originally requested file. As a consequence, TRUMANBOX returns data the attacker most probably does not expect. If the attacker examines the returned data,

e.g. by calculating the hash value of the received content and comparing it with the expected value, he will notice that the received data is not what he requested. In case the attacker detects that TRUMANBOX is in place, he might simply stop his actions on the compromised system so that we would not be able to gather any more information. One possible countermeasure, at least during malware investigation, is to set up a communication channel between TRUMANBOX and the system performing malware samples. Whenever a sample does not perform any activity after a reply of TRUMANBOX, the malware is restarted and TRUMANBOX acts in a different way, e.g. does not grant access to an FTP server this time.

In summary, the advantage of full emulation is clearly that we do not need a connection to the Internet. Hence, we could also use this mode in an offline analysis environment.

Half Proxy Mode All services running within TRUMANBOX are set up in their basic configuration. Thus, server responses will correspond to the defaults which most likely differ from the original server a certain connection request was aiming to contact. In full emulation mode, an attacker might therefore try to fingerprint the network environment. To overcome this problem, we introduced the half proxy mode by substituting certain service features using access to the Internet. For example, if there is no initial payload on the client side, we establish a connection to the Internet trying to fetch the banner from the original server. In case of FTP, the last step is extended to also test if anonymous login is provided. The resulting information is then used for protocol identification.

As a result, an attacker is not able to identify TRUMANBOX by sending requests to unreachable servers since TRUMANBOX contacts the originally targeted servers and acts exactly the same way. Yet, detecting TRUMANBOX by investigating received responses is still possible because the current implementation of TRUMANBOX does not forward the original file content to the attacker. However, future versions of TRUMANBOX will support this feature so that attackers always get the expected file content and investigations of that data will not reveal any information whether TRUMANBOX is in place.

3.3 Extensions to Half Proxy Mode

The following functionality is mainly based on half proxy mode. These functions show how incremental improvements can be done in certain situations to improve the "reality" of the emulation.

Replaying FTP and SMTP Banners: Assuming RFC conform behavior, FTP or SMTP are SFS protocols where payloads are sent first from server to client. Thus, we already fetch banner or welcome messages from the original server during the protocol identification by payload. All gathered banners are stored locally in a cache (a plain text file named with the IP address and the TCP port number of the corresponding server). By checking the cache before contacting the original server, we avoid unnecessary data traffic.

Emulation Efforts on FTP and HTTP: FTP or HTTP connections are particularly challenging since the corresponding servers usually provide a complex and unknown directory structure that the client may wish to access. For example, a client may try to access `http://example.org/path/to/file`. We handle this as follows: After accepting the incoming connection, we identify the protocol by matching the `GET /` pattern in the first payload sent by the client. After parsing the URL, we on the fly create a filesystem structure in our webserver's base directory according to the request we just received. We do not aim at serving the content the client expects in the requested path, even though this could be easily be done in half proxy mode. Our interest is to track what a client is doing. With our approach, we get an overview of the client's behavior on the contacted server. Assuming the client does not send false requests to verify being connected to the server it was intending to connect to, we will learn something about the remote side and the filesystem structure provided there.

FTP User Authentication vs. Anonymous Login: Supporting FTP in our emulation, we need to reason whether to grant anonymous login or rather require a valid username/password combination. In the latter case, we also have to define which credentials are to be accepted and which ones are invalid.

Reviewing our aim to provide an emulation as close to reality (here: the Internet) as possible and also driving a mode that allows us to access the original server, we decide to provide both anonymous login and user authentication, depending on what the original server requires. This is simply done by extending our "fetch the banner" function by a check if, in case of an FTP server, anonymous login is granted. If so, we provide the same and let anonymous login attempts pass. Otherwise, we grant access with any arbitrary login. Even though this might lead to logging a lot of invalid FTP logins we cannot use for further investigation, we hope to also gather valid account data we can use to login ourselves into the original server, for example, to fetch further malicious binaries for analyses. Again, this can be done either manually or in an automated manner by extending our program with functions accordingly. Note that manually processed investigations may be restricted to a certain time slot because logins might expire or get deactivated if the client becomes aware of our interception.

In case of a non-anonymous login, we stick to our approach of not using highly customized configurations but rather extending our payload alteration. Therefore, we just setup our FTP server with one valid login, where the corresponding username and password is known to TRUMANBOX. Any login attempt is then altered to the valid username and password combination. By logging the unmodified login data, we try to gather valid account data for the original server we may use for further investigation. Unfortunately, this approach lacks in handling false login testing, i.e., it is easy for the malware to fingerprint TRUMANBOX by using random combinations of login and password. We can detect such behavior by extending the login payload alteration to recognize if interaction just stops after a successful login and during the next execution of the malware preventing a login to the same server using the same credentials. This idea requires conditionally repeated execution of a malware on the client side.

IRC Session Logging and Emulation: For IRC sessions, issues regarding logging have to be considered. When logging the whole communication the log can become very

long. In particular long sequences of PINGs and PONGs which are often used for alive testing during an IRC session do not improve our analyses. The alternative is to use a whitelist of patterns. Both logging techniques enable us to replay the login process on the original server and thus gather information, for example, about botnets. One possible application is to feed botnet monitoring programs like botspy [9] with the collected information.

3.4 Logging

During the runtime of TRUMANBOX, all incoming payloads and header information are logged to plaintext files named with the IP address and TCP port of the destination target. For this reason, we have to perform outgoing DNS traffic, namely DNS requests, to get this information accurately. If the requested full qualified domain name (FQDN) cannot be resolved, we are faced with a new challenge, but for now we assume that the domain can be resolved. The log files are stored in the subfolders *ftp*, *http*, *irc*, and *smtp* – respectively the protocol of the connection. At the beginning of every new connection attempt, we add a timestamp to the log file that helps us to determine time and date of the logged information. Later, we can use the connection information specified in the filename in combination with the corresponding file content to replay a login or just analyze the traffic directed to that machine.

Provided with the logging structure just outlined, we now have to decide which information to log. To be prepared for different logging strategies, we have implemented a switch deciding to either log everything or just log those information matching certain patterns, where patterns are basically protocol directives we expect to appear together with interesting information, e.g., the patterns `NICK`, `USER`, and `PASS`.

4 Evaluation

We now investigate the practical use of the TRUMANBOX approach.

4.1 Testbed and Evaluation Setting

We set up a testbed to analyze the capability of TRUMANBOX to detect malicious network traffic generated by 300 randomly selected samples taken from the ANUBIS [2, 7] database. Our testbed consists of a Windows 7 system which executes samples in a virtual Windows XP machine and a system running Ubuntu 11.04 which is exclusively used for TRUMANBOX and the local HTTP, FTP, IRC, and SMTP services. The first network interface of TRUMANBOX is connected to the Internet, while the second interface points to the Windows 7 system. The two interfaces of TRUMANBOX are bridged so that all outgoing traffic caused by the ANUBIS samples has to pass TRUMANBOX and can be investigated.

We used this setting to run TRUMANBOX in full emulation and half proxy mode to observe network traffic from malware samples executed on the virtual machine within a two minute lifespan. In the evaluation, we compared the TRUMANBOX logs with the ANUBIS sandbox reports from the database that were created running the malware sample with full Internet access.

4.2 Full Emulation Mode

Our main focus was to evaluate the full emulation mode because it does not require an Internet connection. Interestingly, we were able to show that TRUMANBOX is still able to collect valuable information from the network traffic to improve analysis of malware. We ran the sample set of 300 samples from the ANUBIS database in our virtual Windows XP machine. Table 2 outlines the results we observed.

Table 2: Evaluation results of the 300 investigated ANUBIS samples

Positive network traffic	**#**
TRUMANBOX log and ANUBIS report cover the same information	98
Additional HTTP information in TRUMANBOX log	12
Additional IRC information in TRUMANBOX log	4
Additional FTP information in TRUMANBOX log	20
Total positive	134
Negative network traffic	**#**
No IRC but HTTP traffic in TRUMANBOX log	1
No FTP but HTTP and IRC traffic in TRUMANBOX log	3
Undetected/Invalid IRC format	9
Less HTTP information in TRUMANBOX log	7
Total negative	20
Sum positive and negative	154
Overall	**#**
Number of samples generating network traffic	154
Number of samples not generating network traffic	146
Overall number of samples	300

Out of the 300 randomly selected samples, merely 154 samples generated network traffic at all. The other 146 samples did not attempt to perform any network activities. We can only guess that either these samples do not generate network traffic at all or the network functions were just not triggered for unknown reasons.

When taking a look at the TRUMANBOX log files of the remaining 154 samples performing network activities, our tool was able to collect the same information which is stated in the ANUBIS reports for 98 samples. These samples cover various HTTP requests, FTP connections, and IRC traffic. However, the TRUMANBOX log files do not include one particular type of data: since we do not access the Internet in full emulation mode and thus cannot fetch information from the originally targeted servers, all IRC channel topics are unknown to TRUMANBOX. Channel topics of command and control servers often include URLs which are downloaded once malware joins the channel. The majority of ANUBIS reports cover network traffic of such HTTP and FTP downloads. In full emulation, TRUMANBOX certainly is not able to provide these download URLs to the investigated samples, thus the log files do not include this type of traffic. Nevertheless, all other network activities targeting HTTP, FTP, and IRC were logged accordingly and resemble the data in the ANUBIS reports.

One of the 154 samples did not establish an IRC connection next to HTTP GET requests although the ANUBIS report included information about a command and control server. Furthermore, three of the 154 samples did not establish FTP connections next to HTTP and IRC traffic, although the ANUBIS reports covered details about FTP connections. We can only guess that the IRC and FTP functions were not triggered for unknown reasons.

Furthermore, nine samples did not use the proper IRC protocol so that TRUMANBOX either did not redirect the connection to the local running IRC server or the IRC server refused the connection. Some of the samples used commands like

```
CK [DE|WinXP|15718] 0 * :mIRC 6.17
```

instead of `NICK` and `USER`, others simply did not send the `NICK` command. We speculate that the malware samples perform this non-standard conform communication in order to bypass protocol-based detection of malware communication.

Seven of the 154 samples generated fewer HTTP requests as stated in the ANUBIS reports. On the one hand, we guess that some of the HTTP GET requests were just not triggered for unknown reasons. On the other hand, some malware expects the downloaded content to be valid, e.g., executable files. However, we do not provide the malware with the exact data it expects since we would need to connect to the Internet to get those files. If a malware tries to download a file, performs a check, and notices that it did not get the proper file, it might stop performing further HTTP requests. Yet, some of the investigated malware samples exactly did the opposite and tried to connect to various other download locations. As a result, 12 of the 154 ANUBIS samples performed more requests than stated in the ANUBIS reports, and we were able to extract information about these backup web and command and control servers in an automated way. This includes details about several other domains providing the same malicious file, but also information about completely different filenames and URLs.

Furthermore, four of the 154 samples revealed *more* details about IRC connections compared to the corresponding ANUBIS reports. The TRUMANBOX log files provide more information about IRC channels (because of network downtime of the originally targeted server) and commands performed by the malware. Even a connection to a previously unknown command and control server was established.

Overall, we achieved the best results with the FTP protocol. 20 out of the 154 TRUMANBOX log files provided more information about FTP connections than the corresponding ANUBIS reports. In the majority of cases, more uploads, downloads, and other file operations were logged by TRUMANBOX. Some samples sent the wrong FTP credentials to the target server, so ANUBIS was not able to log any FTP traffic except the unsuccessful login attempt. Since TRUMANBOX redirects the FTP connection to our local FTP server which accepts all login information regardless of the credentials sent by the malware, the samples were able to log in and perform various file operations. A further advantage of running a local FTP server lies in the fact that we can access and analyze all uploaded files in the FTP directory without extracting the file content from the network traffic logs.

Unfortunately, connection attempts to SMTP servers were not initiated by any of the 300 randomly selected samples, so neither ANUBIS reports nor TRUMANBOX log files

included any details about SMTP connections. However, we expect to achieve similar results compared to the other network protocols.

4.3 Half Proxy Mode

In half proxy mode, TRUMANBOX redirects all traffic coming from the virtual machine to our local services, whereas TRUMANBOX itself tries to connect to the originally targeted servers to fetch banners and other information. Thus, the malware receives exactly the same information as it would get when it is directly connected to the Internet. As a consequence, TRUMANBOX log files contain exactly the information which is already covered by the ANUBIS reports. However, in some cases, TRUMANBOX log files include even more information, for example, when a command and control server is not reachable anymore. Using ANUBIS, the malware sample does not perform any communication when the connection attempt failed. In contrast, the connection is redirected to our local running IRC service when using TRUMANBOX, which is available the whole time. Hence, the malware performs its regular communication with our service which is then logged to the TRUMANBOX report. We found that operating TRUMANBOX in half proxy mode results in receiving at least the same information we would obtain by using a "sandbox" environment like ANUBIS and CWSANDBOX. In special cases like downtimes of command and control servers or invalid FTP credentials sent by the malware, TRUMANBOX is capable of obtaining more information about the performed steps of a malware sample.

5 Conclusion

We presented TRUMANBOX, an approach to improve dynamic malware analysis by emulating the Internet. We presented the design and implementation of TRUMANBOX and performed an evaluation. In conclusion, the full emulation mode offered by TRUMANBOX provides us with the opportunity to obtain all information retrieved by "sandbox" environments in most cases. Due to the fact that we do not access the Internet at all, some of the reports, however, might only include parts of the traffic logged by tools accessing the Internet. Yet, TRUMANBOX is effective when the originally targeted servers are unavailable or when malware samples perform failed login attempts.

There are many possible improvements that may be implemented in the freely available source code of TRUMANBOX [4]. For example, full emulation mode can be improved by making it less static, e.g., by randomizing server responses. Also, to thwart fingerprinting attempts by random login and password guessing, a mechanism for conditionally repeated execution of a malware on the client side would be useful. However, this would require a communication channel to the client side. Also, to improve the analysis of IRC channels, TRUMANBOX might send self-generated `NICK random` commands when it detects the lack of `NICK` in the first payload sent to the IRC server.

Acknowledgments

We wish to thank Truman Burbank and Peter Weir for their previous work that strongly influenced this paper.

This work has been supported by the the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant 315-43-02/2-005-WFBO-009) and the Federal Ministry of Education and Research (Grant 01BY1020 – MobWorm). We also thank the anonymous reviewers for their valuable insights and comments.

References

1. Paul Barford and Mike Blodgett. Toward botnet mesocosms. In *Proceedings of the USENIX First Workshop on Hot Topics in Understanding Botnets (HotBots I)*, April 2007.
2. Ulrich Bayer, Andreas Moser, Christopher Krügel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, 2006.
3. G. Chamales. The Honeywall CD-ROM. *IEEE Security & Privacy Magazine*, 2(2):77–79, 2004.
4. Christian Gorecki. TrumanBox – Internet Emulation. `http://trumanbox.s6y.org`, 2011.
5. Flux Group. Emulab: Network emulation testbed home, Accessed: July 2007. Internet: `http://www.emulab.net`.
6. Thomas Hungenberg and Matthias Eckert. INetSim – Internet Simulation. `http://www.inetsim.org`, 2011.
7. International Secure Systems Lab. Anubis: Analyzing unknown binaries. `http://anubis.iseclab.org`, 2011.
8. James Morris. libipq: iptables userspace packet queuing library, Accessed: July 2007. Internet: `https://svn.netfilter.org/netfilter/trunk/iptables/libipq`.
9. Claus R. F. Overbeck. Botspy – efficient observation of botnets. Presentation at Hack.lu, October 2007.
10. The Honeynet Project. *Know Your Enemy: Learning About Security Threats*. Addison-Wesley, 2nd edition, 2004.
11. Niels Provos. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, 2004.
12. Niels Provos and Thorsten Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley, 1st edition, 2007.
13. Carsten Willems, Thorsten Holz, and Felix C. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy Magazine*, 5(2):32–39, March-April 2007.