Revealing Middlebox Interference with Tracebox

Gregory Detal, Benjamin Hesmans, Olivier Bonaventure
Université catholique de Louvain
Louvain-la-Neuve – Belgium
firstname.name@uclouvain.be

Yves Vanaubel, Benoit Donnet
Université de Liège
Liège – Belgium
firstname.name@ulg.ac.be

ABSTRACT

Middleboxes such as firewalls, NAT, proxies, or Deep Packet Inspection play an increasingly important role in various types of IP networks, including enterprise and cellular networks. Recent studies have shed the light on their impact on real traffic and the complexity of managing them. Network operators and researchers have few tools to understand the impact of those boxes on any path. In this paper, we propose `tracebox`, an extension to the widely used `traceroute` tool, that is capable of detecting various types of middlebox interference over almost any path. `tracebox` sends IP packets containing TCP segments with different TTL values and analyses the packet encapsulated in the returned ICMP messages. Further, as recent routers quote, in the ICMP message, the entire IP packet that they received, `tracebox` is able to detect any modification performed by upstream middleboxes. In addition, `tracebox` can often pinpoint the network hop where the middlebox interference occurs. We evaluate `tracebox` with measurements performed on PlanetLab nodes. Our analysis reveals various types of middleboxes that were not expected on such an experimental testbed supposed to be connected to the Internet without any restriction.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network Monitoring

Keywords

Network Discovery, Middleboxes, tracebox

1. INTRODUCTION

The TCP/IP architecture was designed to follow the end-to-end principle. A network is assumed to contain hosts implementing the transport and application protocols, routers implementing the network layer and processing packets, switches operating in the datalink layer, etc. This textbook description does not apply anymore to a wide range of networks. Enterprise networks, WiFi hotspots, and cellular networks often include various types of middleboxes in addition to traditional routers and switches [1]. A *middlebox*, defined as "any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host" [2], manipulates traffic for purposes other than simple packet forwarding. Middleboxes are often deployed for performance or security reasons. Typical middleboxes include Network Address Translators, firewalls, Deep Packet Inspection boxes, transparent proxies, Intrusion Prevention/Detection Systems, etc.

Recent papers have shed the light on the deployment of those middleboxes. For instance, Sherry et al. [1] obtained configurations from 57 enterprise networks and revealed that they can contain as many middleboxes as routers. Wang et al. [3] surveyed 107 cellular networks and found that 82 of them used NATs. Although these middleboxes are supposed to be transparent to the end-user, experience shows that they have a negative impact on the evolvability of the TCP/IP protocol suite [4]. For example, after more that ten years of existence, SCTP [5] is still not widely deployed, partially because many firewalls and NAT may consider SCTP as an unknown protocol and block the corresponding packets. Middleboxes have also heavily influenced the design of Multipath TCP [4, 6].

Despite of their growing importance in handling operational traffic, middleboxes are notoriously difficult and complex to manage [1]. One of the causes of this complexity is the lack of debugging tools that enable operators to understand where and how middleboxes interfere with packets. Many operators rely on `ping`, `traceroute`, and various types of `show` commands to monitor their networks.

In this paper, we propose, validate, and evaluate `tracebox`. `tracebox` is a `traceroute` [7] successor that enables network operators to detect which middleboxes modify packets on almost any path. `tracebox` allows one to easily generate complex probes to send to any destination. By using the quoted packet inside of ICMP replies, it allows to identify various types of packet modifications and can be used to pinpoint where a given modification takes place.

The remainder of this paper is organized as follows: Sec. 2 describes how `tracebox` works and how it is able to identify middleboxes along a path. Sec. 3 analyses three use cases from a deployment of `tracebox` on PlanetLab. Sec. 4 shows how `tracebox` can be used to debug networking issues. Sec. 5 compares `tracebox` regarding state of the art. Finally, Sec. 6 concludes and discusses further work.

IMC'13, October 23–25, 2013, Barcelona, Spain.

http://dx.doi.org/10.1145/2504730.2504757.

2. TRACEBOX

To detect middleboxes, `tracebox` uses the same incremental approach as `traceroute`, i.e., sending probes with increasing TTL values and waiting for ICMP `time-exceeded` replies. While `traceroute` uses this information to detect intermediate routers, `tracebox` uses it to infer the modification applied on a probe by an intermediate middlebox.

`tracebox` brings two important features.

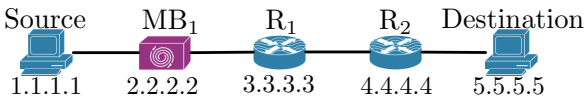
Middleboxes Detection `tracebox` allows one to easily and precisely control all probes sent (IP header, TCP or UDP header, TCP options, payload, etc.). Further, `tracebox` keeps track of each transmitted packet. This permits to compare the quoted packet sent back in an ICMP `time-exceeded` by an intermediate router with the original one. By correlating the different modifications, `tracebox` is able to infer the presence of middleboxes.

Middleboxes Location Using an iterative technique (in the fashion of `traceroute`) to discover middleboxes also allows `tracebox` to approximately locate, on the path, where modifications occurred and so the approximate middleboxes position.

When an IPv4 router receives an IPv4 packet whose TTL is going to expire, it returns an ICMPv4 `time-exceeded` message that contains the offending packet. According to RFC792, the returned ICMP packet should quote the IP header of the original packet and the first 64 bits of the payload of this packet [8]. When the packet contains a TCP segment, these first 64 bits correspond to the source and destination ports and the sequence number. RFC1812 [9] recommended to quote the entire IP packet in the returned ICMP, but this recommendation has only been recently implemented on several major vendors' routers. Discussions with network operators showed that recent routers from Cisco (running IOX), Alcatel Lucent, HP, Linux, and Palo-Alto firewalls return the full IP packet. In the remainder of this paper, we use the term *Full ICMP* to indicate an ICMP message quoting the entire IP packet. We use the term RFC1812-compliant router to indicate a router that returns a *Full ICMP*.

By analyzing the returned quoted packet, `tracebox` is able to detect various modifications performed by middleboxes and routers. This includes changes in the Differentiated Service field and/or the Explicit Congestion Notification bits in the IP header, changes in the IP identification field, packet fragmentation, and changes in the TCP sequence numbers. Further, when `tracebox` receives a *Full ICMP*, it is able to detect more changes such as the TCP acknowledgement number, TCP window, removal/addition of TCP options, payload modifications, etc.

`tracebox` also allows for more complex probing techniques requiring to establish a connection and so multiple probes to be sent, e.g., to detect segment coalescing/splitting, Application-level Gateways, etc. In this case `tracebox` works in two phases: the *detection* and the *probing* phases. During the detection phase, `tracebox` sends probes by iteratively increasing the TTL until it reaches the destination. This phase allows `tracebox` to identify RFC1812-compliant routers. During the probing phase, `tracebox` sends additional probes with TTL values corresponding to the previously discovered RFC1812-compliant routers. This strategy allows `tracebox` to reduce its overhead by limiting the number of probes sent.



(a) topology

```
# tracebox -p 'IP / TCP / mss(9000)' -n 5.5.5.5
tracebox to 5.5.5.5 (5.5.5.5): 30 hops max
1: 3.3.3.3 TCP::SequenceNumber
2: 4.4.4.4 IP::TTL IP::CheckSum TCP::CheckSum TCP::SequenceNumber
           TCPOptionMaxSegSize::MaxSegSize
3: 5.5.5.5
```

(b) output

Figure 1: `tracebox` example

Fig. 1(a) shows a simple network, where MB_1 is a middlebox that changes the TCP sequence number and the MSS size in the TCP MSS option but that does not decrement the TTL. R_1 is an old router while R_2 is a RFC1812-compliant router. The server always answer with a TCP reset. The output of running `tracebox` between "Source" and "Destination" is given by Fig. 1(b). The output shows that `tracebox` is able to effectively detect the middlebox interference but it may occur at a downstream hop. Indeed, as R_1 does not reply with a *Full ICMP*, `tracebox` can only detect the TCP sequence change when analyzing the reply of R_1. Nevertheless, when receiving the *Full ICMP* message from R_2, that contains the complete IP and TCP header, `tracebox` is able to detect that a TCP option has been changed upstream of R_2. At the second hop, `tracebox` shows additional modifications on top of the expected ones. The TTL and IP checksum are modified by each router and the TCP checksum modification results from the modification of the header.

The detection of middleboxes depends on the reception of ICMP messages. If the router downstream of a middlebox does not reply with an ICMP message, `tracebox` will only be able to detect the change at a downstream hop similarly as the above example. Another limitation is that if the server does not reply with an ICMP (as in Fig. 1), then the detection of middleboxes in front of it is impossible.

`tracebox` is implemented in C++ in about 2,000 lines of code and embeds LUA [10] bindings to allow a flexible description of the probes as well to ease the development of more complex middlebox detection scripts. `tracebox` aims at providing the user with a simple and flexible way of defining probes without requiring a lot of lines of code. `tracebox` indeed allows to use a single line to define a probe (see as example the argument `-p` of `tracebox` in Fig. 1(b)) similarly to Scapy [11]. `tracebox` provides a complete API to easily define IPv4/IPv6 as well as TCP, UDP, ICMP headers and options on top of a raw payload. Several LUA scripts are already available and allows one to detect various types of middleboxes from Application-level Gateways to HTTP proxies. It is open-source and publicly available [12].

To verify the ability of `tracebox` to detect various types of middlebox interference, we developed several Click elements [13] modeling middleboxes. We wrote Click elements that modify various fields of the IP or TCP header, elements that add/remove/modify TCP options and elements that coalesce or split TCP segments. These elements have been included in a python library [14] that allows to easily describe a set of middleboxes and that generates the corresponding Click configuration. This library is used as unit

tests to validate each new version of `tracebox`.

3. VALIDATION & USE CASES

In this section, we validate and demonstrate the usefulness of `tracebox` based on three use cases. We first explain how we deploy `tracebox` on the PlanetLab testbed (Sec. 3.1), next we asses the coverage of `tracebox` (Sec. 3.2) and finally discuss our use cases (Sec. 3.3, and 3.4).

3.1 PlanetLab Deployment

We deployed `tracebox` on PlanetLab, using 72 machines as vantage points (VPs). Each VP had a target list of 5,000 items build with the top 5,000 Alexa web sites. Each VP used a shuffled version of the target list. DNS resolution was not done before running `tracebox`. This means that, if each VP uses the same list of destination names, each VP potentially contacted a different IP address for a given web site due to the presence of load balancing or Content Distribution Networks. Our dataset was collected during one week starting on April 17th, 2013.

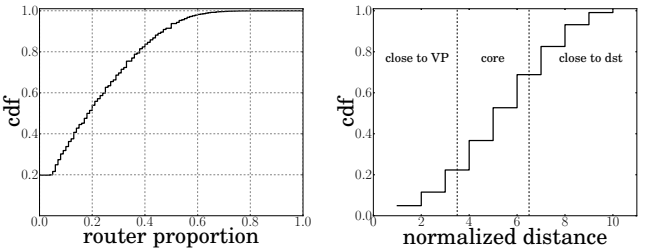
In this short paper, we focus on analyzing some interferences between middleboxes and TCP. In theory, PlanetLab is not the best place to study middleboxes because PlanetLab nodes are mainly installed in research labs with unrestricted Internet access. Surprisingly, we noticed that seven VPs, from the 72 considered for the use cases, automatically removed or changed TCP options at the very first hop. They replaced the Multipath TCP [6], MD5 [15], and Window Scale [16] options with NOP and changed the value of the MSS option. We also found that two VPs always change the TCP Sequence number.

3.2 RFC1812-compliant routers

`tracebox` keeps track of each original packet sent and makes a comparison with the quoted IP packet when the ICMP `time-exceeded` message is received. Further, `tracebox` can potentially detect more middleboxes when routers return a *Full ICMP* packet. `tracebox`'s utility clearly increases with the number of RFC1812-compliant routers. Fig. 2 provides an insight of the proportion of RFC1812-compliant routers and their locations.

In particular, Fig. 2(a) gives the proportion of RFC1812-compliant routers (the horizontal axis) as a CDF. A value of 0, on the horizontal axis, corresponds to paths that contained no RFC1812-compliant router. On the other hand, a value of 1 corresponds to paths made only of RFC1812-compliant routers. Looking at Fig. 2(a), we observe that, in 80% of the cases, a path contains at least one router that replies with a *Full ICMP*. In other words, `tracebox` has the potential to reveal more middleboxes in 80% of the cases.

Fig. 2(b) estimates the position of the RFC1812-compliant routers in the probed paths. It provides the distance between the VP and the RFC1812-compliant routers on a given path. Note that, on Fig. 2(b), the X-Axis (i.e., the distance from the VPs) has been normalized between 1 and 10. Distances between 1 and 3 refer to routers close to the VP, 4 and 6 refer to the Internet core while, finally, distances between 7 and 10 refer to routers closer to the `tracebox` targets. The widespread deployment of RFC1812-compliant routers in the Internet core is of the highest importance since `tracebox` will be able to use these routers as "mirrors" to observe the middlebox interferences occurring in the access network [3].



(a) Proportion of RFC1812-compliant routers on a path (b) Normalized distance from VP to RFC1812-compliant router

Figure 2: RFC1812-compliant routers

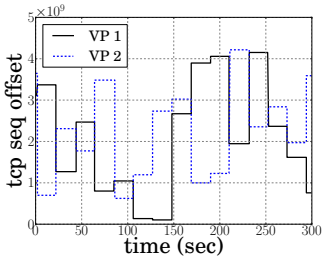


Figure 3: Time evolution of the TCP sequence number offset introduced by middleboxes

Fig. 2(b) shows that for 22% of the paths, the RFC1812-compliant routers are close to the VP. This is approximatively the same proportion for routers close to `tracebox` targets. However, the majority of routers sending back a *Full ICMP* are located in the network core.

3.3 TCP Sequence Number Interference

The TCP sequence number is not supposed to be modified by intermediate routers. Still, previous measurements [4] showed that some middleboxes change sequence and acknowledgement numbers in the processed TCP segments. As the sequence number is within the first 64 bits of the TCP header, `tracebox` can detect its interference even though there are none RFC1812-compliant routers.

We analyze the output of `tracebox` from the 72 VPs. Our measurements reveal that two VPs always modify the TCP sequence numbers. The position of the responsible middlebox is close to the VP, respectively the first and third hop. We suspect that the middlebox randomizes the TCP sequence number to fix a bug in old TCP/IP stacks where the *Initial Sequence Number* (ISN) was predictable [17].

When used on a path that includes such a middlebox, `tracebox` can provide additional information about the sequence number randomization. Depending on the type of middlebox and the state it maintains, the randomization function can differ. To analyze it, we performed two experiments. First, we generated SYN probes with the same destination (IP address and port), the same sequence number, and different source ports. `tracebox` revealed that the middlebox modified all TCP sequence numbers as expected. A closer look at the modified sequence numbers revealed that the difference between the ISN of the probe and the randomized sequence number can be as small as 14510 and as large as 4294858380 (which corresponds to a negative difference

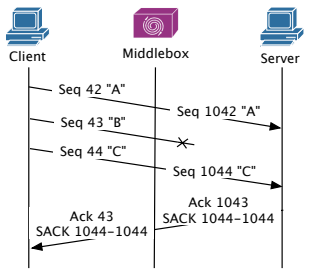


Figure 4: Example of invalid SACK blocks generated due to a middlebox.

of 108916 when the 32 bits sequence number space wrap). Our measurements show that these differences appear to be uniformly distributed for the different source ports.

For our second experiment, we used `tracebox` to verify how the randomization evolves over time. For this, we sent SYN probes using the same 5-tuple and different ISN during five minutes and evaluated the evolution of the TCP sequence number modifications. Fig. 3 shows the offset between the sent ISN and the randomized one for two different 5-tuples. `tracebox` reveals that the two middleboxes seem to change their randomization approximatively every 20 seconds. This suggests stateful middleboxes.

As explained by Honda et al. [4], changing the TCP sequence numbers has an impact on the TCP protocol evolvability. Unfortunately, it has also an impact on the utilization of widely deployed TCP extensions. Consider the TCP *Selective Acknowledgement* (SACK) option [18]. This TCP option improves the ability of TCP to recover from losses. One would expect that a middlebox changing the TCP sequence number would also update the sequence numbers reported inside TCP options. This is unfortunately not true, almost 18 years after the publication of the RFC [18]. We used `tracebox` to open a TCP connection with the SACK extension and immediately send SACK blocks. `tracebox` reveals that the middlebox changes the sequence number but does not modify the sequence number contained in the SACK block.

Fig. 4 shows the behavior of such a middlebox on the TCP sequence number and SACK blocks. In this scenario, the middlebox increases the TCP sequence number by 1,000 bytes causing the client to receive a SACK block that corresponds to a sequence number that it has not yet transmitted. This SACK block is invalid, but the acknowledgement is valid and correct. For the receiver, it may not be easy to always detect that the SACK information is invalid. The receiver may detect that the SACK blocks are out of the window, but the initial change may be small enough to carry SACK blocks that are inside the window.

If we know that a SACK block is invalid, algorithms that use SACK should understand that the SACK option does not give more information than a simple acknowledgment. In this view, such algorithms should have at least the same performance as they would have if SACK was not used at all. Unfortunately, this is not the case as the Linux TCP stack does not consider duplicate acknowledgment when SACK is enabled. When the offset is small the SACK blocks are potentially in-window. In this case the Linux TCP stack reacts correctly. However, when the SACK blocks are out-of-window then the TCP stack has to wait for a complete RTO instead of doing fast-retransmit. We performed a small measurement in a controlled environment and observed up to a 50% drop in performance with a large offset [19].

3.4 TCP MSS Option Interference

Our third use case for `tracebox` concerns middleboxes that modify the TCP MSS option. This TCP option is used in the SYN and SYN+ACK segments to specify the largest TCP segment that a host sending the option can process. In an Internet that respects the end-to-end principle, this option should never be modified. In the current Internet, this is unfortunately not the case. The main motivation for changing the TCP MSS option on middleboxes is probably to fix some issues caused by other middleboxes with Path MTU Discovery [20]. On top of changing the MSS option, we also discovered middleboxes, in a couple of ISPs, that add the option if it is missing.

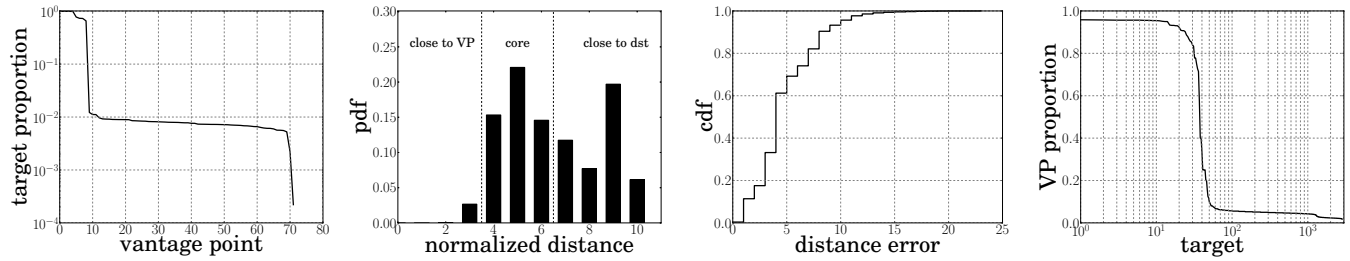
Path MTU Discovery is a technique that allows a host to dynamically discover the largest segment it can send without causing IP fragmentation on each TCP connection. For that, each host sends large segments inside packets with the `Don't Fragment` bit set. If a router needs to fragment the packet, it returns an ICMP `destination-unreachable` (with code "Packet fragmentation is required but the 'don't fragment' flag is on") back to the source and the source updates its segment size. Unfortunately, some routers do not return such ICMP messages [21] and some middleboxes (e.g., NAT boxes and firewalls) do not correctly forward the received ICMP message to the correct source. MSS clamping mitigates this problem by configuring middleboxes to decrease the size reported in the MSS option to a smaller MSS that should not cause fragmentation.

We use our dataset to identify middleboxes that modify the MSS option in SYN segments. Fig. 5(a) provides, for each VP (the horizontal axis), the proportion of paths (the vertical axis, in log-scale) where the MSS option has been changed. We see that a few VPs encountered at least one MSS modification on nearly all paths while, for the vast majority of VPs, the modification is observed in only a couple of paths. We decided to remove those VPs from our data set for further analyses, meaning that only 65 VPs were finally considered for the use case.

Similarly to Fig. 5(a), Fig. 5(d) provides, for each target, the proportion of paths affected by an MSS modification. We see about ten targets that have a middlebox, probably their firewall or load balancer, always changing the MSS option. In the same fashion as the VPs that changed the MSS option, they also removed the Multipath TCP, MD5 and Window Scale options.

Fig. 5(b) indicates where, in the network, the MSS option is modified. In the fashion of Fig. 2(b), the distance from VP has been normalized between 1 and 10, leading to the rise of three network regions (i.e., close to VP, core, and close to targets). As shown by Fig. 5(b), `tracebox` can detect the MSS modification very close to the source (2.7% of the cases) while this detection mostly occurs in the network core (52% of the cases).

Remind that this distance does not indicate precisely where is actually located the middlebox responsible for the MSS modification. Rather, it gives the position of the router that has returned a *Full ICMP* and, in this ICMP packet, the quoted TCP segment revealed a modification of the MSS field. Actually, the middlebox should be somewhere between



(a) VPs proportion modifying MSS (b) Location (c) Location error (d) Targets proportion observing an MSS modification

Figure 5: MSS option modification

this position and the previous router (on that path) that has also returned a *Full ICMP* (or the VP if it was the very first *Full ICMP* on that path).

Fig. 5(c) refines our location of MSS modification by taking this aspect (i.e., the middlebox is somewhere on the path between the modification detection and the previous RFC1812-compliant router) into account. It gives thus an estimation of middlebox location error. This error is simply obtained by subtracting the distance at which `tracebox` reveals the modification and the distance at which the previous RFC1812-compliant router was detected by `tracebox` on that path. Obviously, lower the error, more accurate the location given in Fig. 5(b). On Fig. 5(c), we see that in 61% of the cases, the location estimation error is below (or equal to) four hops. All errors above 13 hops, that represents the length of around 60% of the paths, are uncommon (less than 1% each).

4. DISCUSSION

In Sec. 3, we showed that `tracebox` can provide a useful insight on known middleboxes interference. We believe that `tracebox` will also be very useful for network operators who have to debug strange networking problems involving middleboxes. While analyzing the data collected during our measurement campaign (see Sec. 3.1), we identified several strange middlebox behaviors we briefly explain in this section. We also discuss how `tracebox` can be used to reveal the presence of proxies and network address translators (NATs).

4.1 Unexpected Interference

We performed some tests with `tracebox` to verify whether the recently proposed Multipath TCP [6] option could be safely used over the Internet. This is similar to the unknown option test performed by Honda et al. [4]. However, on the contrary to Honda et al., `tracebox` allows one to probe a large number of destinations. To our surprise, when running the tests, `tracebox` identified about ten Multipath TCP servers based on the TCP option they returned. One of those server, `www.baidu.com`, belongs to the top 5 Alexa. All these servers where located inside China. A closer look at these options revealed that these servers (or their load balancers) simply echo a received unknown TCP option in the SYN+ACK. This is clearly an incorrect TCP implementation.

4.2 Proxy Detection

`tracebox` can also be used to detect TCP proxies. To be able to detect a TCP proxy, `tracebox` must be able to send TCP segments that are intercepted by the proxy and other packets that are forwarded beyond it. HTTP proxies are frequently used in cellular and enterprise networks [3]. Some of them are configured to transparently proxy all TCP connections on port 80. To test the ability of detecting proxies with `tracebox`, we used a script that sends a SYN probe to port 80 and, then, to port 21. If there is an HTTP proxy on the path, it should intercept the SYN probe on port 80 while ignoring the SYN on port 21. We next analyze the ICMP messages returned.

From our simple PlanetLab deployment, we identified two oddities. First, we found an HTTP proxy or more probably an IDS within a National Research Network (SUNET) as it only operated for a few destinations and that the path for port 80 was shorter than for port 21. Second, and more disturbing, `tracebox` identified that several destinations where behind a proxy whose configuration, inferred from the returned ICMP messages, resulted in a forwarding loop for probes that are not HTTP. We observed that the SYN probe on port 21, after reaching the supposed proxy, bounced from one router to the other in a loop as `tracebox` received ICMP replies from one router then another alternatively.

4.3 NAT Detection

NATs are probably the most widely deployed middleboxes. Detecting them by using `tracebox` would likely be useful for network operators. However, in addition to changing addresses and port numbers of the packets that they forward, NATs often also change back the returned ICMP message and the quoted packet. This implies that, when inspecting the received ICMP message, `tracebox` would not be able to detect the modification.

This does not prevent `tracebox` from detecting many NATs. Indeed, most NATs implement *Application-level Gateways* (ALGs) [22] for protocols such as FTP. Such an ALG modifies the payload of forwarded packets that contain the `PORT` command on the `ftp-control` connection. `tracebox` can detect these ALGs by noting that they do not translate the quoted packet in the returned ICMP messages. This detection is written as a simple script (shown in Fig 6) that interacts with `tracebox`. It builds and sends a SYN for the FTP port number and, then, waits for the SYN+ACK. The script makes sure that the SYN+ACK is not handled by the TCP stack of the host by configuring the local firewall (using the filter functionality, shown at line 7, of `tracebox` that configures `iptables` on Linux and `ipfw` on Mac OS X). It then sends a valid segment with the `PORT` command and the encoded IP address and port number as payload. `tracebox` then compares the transmitted packet with the

```
1  -- NAT FTP detection
2  -- To run with: tracebox -s <script> <ftp_server>
3  -- Build the initial SYN (dest is passed to tracebox)
4  syn = IP / tcp{dst=21}
5  -- Avoid the host's stack to reply with a reset
6  fp = filter(syn)
7  synack = tracebox(syn)
8  if not synack then
9          print("Server did not reply...")
10         fp:close()
11         return
12 end
13 -- Check if SYN+ACK flags are present
14 if synack:tcp():getflags() ~= 18 then
15         print("Server does not seems to be a FTP server")
16         fp:close()
17         return
18 end
19 -- Build the PORT probe
20 ip_port = syn:source():gsub("%.", ",")
21 data = IP / tcp{src=syn:tcp():getsource(), dst=21,
22         seq=syn:tcp():getseq()+1,
23         ack=synack:tcp():getseq()+1, flags=16} /
24         raw('PORT '.. ip_port .. ',189,68\r\n')
25 -- Send probe and allow cb to be called for each reply
26 function cb(ttl, rip, pkt, reply, mods)
27         if mods and mods:__tostring():find("Raw") then
28                 print("There is a NAT before " .. rip)
29                 return 1
30         end
31 end
32 tracebox(data, {callback = "cb"})
33 fp:close()
```

Figure 6: Sample script to detect a NAT FTP.

quoted packet returned inside an ICMP message by an RFC1812-compliant router and stores the modification applied to the packet. If a change occurs and a callback function has been passed as argument, `tracebox` triggers the callback function. In Fig 6, the callback `cb` checks whether there has been a payload modification. If it is the case a message showing the approximate position of the ALG on the path is printed (see line 29).

5. RELATED WORK

Since the end of the nineties, the Internet topology discovery has been extensively studied [23, 24]. In particular, `traceroute` [7] has been used for revealing IP interfaces along the path between a source and a destination. Since then, `traceroute` has been extended in order to mitigate its intrinsic limitations. From simple extensions (i.e., the types of probes sent [25, 26]) to much more developed modifications. For instance, `traceroute` has been improved to face load balancing [27] or the reverse path [28]. Its probing speed and efficiency has also been investigated [29, 30, 31].

To the best of our knowledge, none of the available `traceroute` extensions allows one to reveal middlebox interference along real Internet paths as `tracebox` does.

Medina et al. [21] report one of the first detailed analysis of the interactions between transport protocols and middleboxes. They rely on active probing with `tbit` and contact various web servers to detect whether Explicit Congestion Notification (ECN) [32], IP options, and TCP options can be safely used. The `TCPExposure` software developed by Honda et al. [4] is closest to `tracebox`. It also uses specially crafted packets to test for middlebox interference.

Wang et al. [3] analyzed the impact of middleboxes in hundreds of cellular networks. This study revealed various types of packet modifications. These three tools provide great results, but they are limited to specific paths as both ends of the path must be under control. This is a limitation since some middleboxes are configured to only process the packets sent to specific destination or ports. On the contrary, `tracebox` does not require any cooperation with the service. It allows one to detect middleboxes on any path, i.e., between a source and any destination. Our measurements reveal middleboxes that are close to the clients but also close to the server.

Sherry et al. [1] have relied on network configuration files to show the widespread deployment of middleboxes. Still, their study does not reveal the impact of these middleboxes on actual packets.

6. CONCLUSION

Middleboxes are becoming more and more popular in various types of networks (enterprise, cellular network, etc.). Those middleboxes are supposed to be transparent to users. It has been shown that they frequently modify packets traversing them, sometimes making protocols useless. Further, due to the lack of efficient and easy-to-use debugging tools, middleboxes are difficult to manage.

This is exactly what we tackled in this paper by proposing, discussing, and evaluating `tracebox`. `tracebox` is a new extension to `traceroute` that allows one to reveal the presence of middleboxes along a path. It detects various types of packet modifications and can be used to locate where those modifications occur. We deployed it on the PlanetLab testbed and demonstrated its capabilities by discussing several use cases. `tracebox` is open-source and publicly available [12].

`tracebox` opens new directions to allow researchers to better understand the deployment of middleboxes in the global Internet. In the coming months, we plan to perform large-scale measurement campaigns to analyze in more details middlebox interferences in IPv4 and IPv6 networks. `tracebox` could also be extended to fingerprint specific middleboxes.

Acknowledgments

We are grateful to the anonymous reviewers for their feedback. We would also like to thank Randy Bush, Matsuzaki Yoshinobu, Marc Neuckens, Pierre Reinbold, Bruno Delcourt and Claire Delcourt for assistance in understanding the middleboxes present in their networks.

This work is partially funded by the European Commission funded mPlane (ICT-318627) and CHANGE (INFSO-ICT-257422) projects and the BESTCOM IAP.

7. REFERENCES

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, August 2012.

[2] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," Internet Engineering Task Force, RFC 3234, February 2002.

[3] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, "An untold story of middleboxes in cellular networks," in *Proc. ACM SIGCOMM*, August 2011.

[4] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP," in *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, November 2011.

[5] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream control transmission protocol," Internet Engineering Task Force, RFC 2960, October 2000.

[6] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," Internet Engineering Task Force, RFC 6824, January 2013.

[7] V. Jacobson et al., "traceroute," UNIX, man page, 1989, see source code: ftp://ftp.ee.lbl.gov/traceroute.tar.gz.

[8] J. Postel, "Internet control message protocol," Internet Engineering Task Force, RFC 792, September 1981.

[9] F. Baker, "Requirements for IP version 4 routers," Internet Engineering Task Force, RFC 1812, June 1995.

[10] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes, "LUA, an extensible extension language," *Software: Pactice & Experience*, vol. 26, no. 6, pp. 635–652, June 1996.

[11] P. Biondi, "Scapy," see http://www.secdev.org/projects/scapy/.

[12] G. Detal, "`tracebox`," July 2013, see http://www.tracebox.org.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, August 2000.

[14] B. Hesmans, "Mbclick," July 2013, see https://bitbucket.org/bhesmans/mbclick.

[15] A. Heffernan, "Protection of BGP sessions via the TCP MD5 signature option," Internet Engineering Task Force, RFC 2385, August 1998.

[16] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," Internet Engineering Task Force, RFC 1323, May 1992.

[17] Microsoft, "Patch available to improve TCP initial sequence number randomness," Microsoft, Microsoft Security Bulletin MS99-066, October 1999, see http://technet.microsoft.com/en-us/security/bulletin/ms99-046.

[18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," Internet Engineering Task Force, RFC 2018, October 1996.

[19] C. Paasch, "Presentation ietf 87," July 2013, see http://tools.ietf.org/agenda/87/slides/slides-87-tcpm-11.pdf.

[20] J. Mogul and S. Deering, "Path MTU discovery," Internet Engineering Task Force, RFC 1191, November 1990.

[21] A. Medina, M. Allman, and S. Floyd, "Measuring interactions between transport protocols and middleboxes," in *Proc. ACM SIGCOMM Internet Measurement Conference (IMC)*, October 2004.

[22] P. Srisuresh and M. Holdrege, "IP network address translator (NAT) terminology and considerations," Internet Engineering Task Force, RFC 2663, August 1999.

[23] B. Donnet and T. Friedman, "Internet topology discovery: a survey," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 4, December 2007.

[24] H. Haddadi, G. Iannaccone, A. Moore, R. Mortier, and M. Rio, "Network topologies: Inference, modeling and generation," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 2, pp. 48–69, April 2008.

[25] M. Torren, "tcptraceroute - a traceroute implementation using TCP packets," UNIX, man page, 2001, see source code: http://michael.toren.net/code/tcptraceroute/.

[26] M. Luckie, Y. Hyun, and B. Huffaker, "Traceroute probe methode and forward IP path inference," in *ACM SIGCOMM Internet Measurement Conference (IMC)*, October 2008.

[27] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with Paris traceroute," in *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, October 2006.

[28] E. Katz-Bassett, H. Madhyastha, V. Adhikari, C. Scott, J. Sherry, P. van Wesep, A. Krishnamurthy, and T. Anderson, "Reverse traceroute," in *Proc. USENIX Symposium on Networked Systems Design and Implementations (NSDI)*, June 2010.

[29] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Efficient algorithms for large-scale topology discovery," in *Proc. ACM SIGMETRICS*, June 2005.

[30] R. Beverly, A. Berger, and G. Xie, "Primitives for active Internet topology mapping: Toward high-frequency characterization," in *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, November 2010.

[31] T. Bourgeau and T. Friedman, "Efficient IP-level network topology capture," in *Proc. Passive and Active Measurement Conference (PAM)*, March 2013.

[32] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," Internet Engineering Task Force, RFC 3168, September 2001.