REVCON: Reverse Network Reconnaissance

Security and Network Engineering, University of Amsterdam, The Netherlands

Tuesday 29th October, 2019

Philipp Mieden
philipp.mieden@os3.nl

Paul Dunn
paul.dunn@os3.nl

Axel Koolhaas
axel.koolhaas@os3.nl

Davide Pucci
davide.pucci@os3.nl

***Abstract*—In this paper we present REVCON, a web service to map enterprise networks that are secured by a stateful firewall, using reverse parasitic tracerouting. Reconnaissance is a key aspect in maintaining access to a compromised network, as intelligence gathered in advance about the target network will reduce the amount of noise that an attacker creates during lateral movement. Intermediate nodes along the path through an internal network are of great interest for an attacker, as they forward traffic from various devices. Penetrating these intermediate hops and sniffing traffic will likely yield user credentials, authorization tokens and other confidential information. After being contacted by a victim from the inside of a protected network, the service will inject probing packets into the connection. Analogously to the traceroute concept, these probes start with a low Time To Live (TTL) value and get incremented for each hop, while the service is listening for Internet Control Message Protocol (ICMP) *TTL Expired* error messages from intermediate devices. We demonstrate that this technique is a threat especially for IPv6 networks and that it can be used to gather multiple paths through load balanced network environments.**

***Index Terms*—network, reconnaissance, security, mapping, traceroute, parasitic**

I. Introduction

Corporate networks are protected by a single or multiple firewalls that are usually configured to prevent internal devices from being contacted by the outside world. Retrieving information about the architecture of such networks should be prohibited for unauthorized parties from the outside. However, in order to successfully exchange data, a stateful firewall must allow traffic that belongs to, or is related to, a connection that has been initiated from a client inside the network. This fact can be abused to exfiltrate information about a network, when a victim contacts a service under the attackers control, such as a malicious website.

To avoid being detected by a Security Operations Center (SOC) investigation or intrusion detection system during lateral movement, attackers try to gather as much information upfront as possible in the reconnaissance phase [1]. Intermediate hops inside the target network are attractive targets for further penetration, as they route traffic from the internal network and therefore potentially forward confidential information. Compromising these enables an attacker to capture network traffic and search for authorization data that can be used to move further through the target network.

Due to the limited address space available for IPv4 networks, Network Address Translation (NAT) is commonly used to translate internal IPv4 addresses to public IPv4 addresses. Since the address space for IPv6 does not face such restrictions [2], NAT is not commonly deployed, except as a compatibility layer to IPv4 or for multitenancy architectures. Additionally, rulesets for IPv4 and IPv6 are maintained separately, which increases the chance for configuration discrepancies that can be used to gather information during the reconnaissance phase. Because the ICMPv6 protocol is especially vital for the correct functioning of IPv6 network infrastructure [3] and a powerful way to troubleshoot and debug problems, ICMPv6 traffic is usually less restricted [4] than ICMP, further increasing the attack surface for such types of reconnaissance.

A. Traceroute basics

The traceroute tool is a common network debugging utility used to identify traversed hosts and measure the delta time during their traversal. The Time To Live (TTL) field on the IP packet header provides a mechanism to limit the lifetime of packets sent across the Internet and therefore prevents loops due to routing misconfiguration or convergeance. The value is decremented by each router along the path before forwarding the packet. When reaching zero, the router will drop the packet and reply to the source address of the dropped packet with an ICMP *TTL Expired* error message. [5] Traceroute uses this mechanism to identify traversed hops, by sending UDP probes with an increasing TTL field on the IP packet header, starting with an inital value of one. As intented by the TTL mechanism, this elicits ICMP "time exceeded" replies from the intermediate devices, revealing their own IP address. To determine when the destination has finally been reached, the traceroute tool addresses the probing packet to a non existing port and stops once it receives a "port unreachable" reply from the destination [6].

B. Reverse network mapping

Stateful firewalls prevent attackers who are located outside the target network, to use the traceroute tool for the purpose of mapping an internal network, as they will deny any connections from the outside that have not been initiated from within the network. However, the traceroute concept can be applied in reverse by injecting the probes into a connection that was

previously established from the inside and therefore allowed by the firewall. Effectiveness of this depends on two factors: the probing packets being able to reach the intermediate devices and the ICMP error responses reaching the malicious service.

In order to have the victim contact the service and punch a hole into the firewall, spear phishing could be used, as this is one of the most common entry vectors [7]. Alternatively the probing packets could be injected by an intermediate device acting as a Man in the Middle (MITM) that is located outside the protected network, but along the path to the remote service. If an attacker compromises (or is the owner of) a popular online service, performing this type of attack would be applicable on a large scale.

C. Research question

We've researched the impact of this type of active reconnaissance for both IPv4 and IPv6 networks and compared the differences in behavior, in order to answer the following questions:

1) Does the effectiveness of the attack differ between IPv4 and IPv6 networks?
2) Is it possible to discover multiple paths in load balanced environments using this technique?
3) How can system administrators mitigate this category of attacks?

D. Related work

The methodology of applying reverse tracerouting to gather information about networks protected by firewalls was initially proposed by Goldsmith et al. [5] in 1998. However, this first proof of concept was not using the parasitic approach of using an already established connection, but instead tried to determine the firewalls Access Control List (ACL) content. This introduced the drawback of being easily detectable from the perspective of an intrusion detection system, because the attacker had to actively probe the firewall.

In 2002, Dan Kaminsky published a tool called *paratrace* as part of *Paketto Keiretsu toolkit 1.0* release [8], that introduced the first parasitic version of this attack. In 2007, Michal Zalewski published *0trace* on the *BuqTraq* Mailing list [9], a proof on concept tool for applying the parasitic approach from a man in the middle perspective. Unfortunately, none of the above proof of concepts seem to be connected to an academic paper. More recent research from Ivan Morandi used this technique for debugging purposes, such as performance analysis, and not with malicious intent [10] [11]. The implications of this type of reconnaissance have not been academically evaluated for IPv6 networks.

II. Methodology

To answer the research questions we've setup a lab environment that models a load balanced enterprise network, protected by a firewall. For carrying out the attack we've implemented a web service written in Golang, that traces the client with explicit consent and displays the results to him. In order to estimate the impact on networks outside of the lab environment, a case study was launched that offers interested network administrators the possibility to test their networks for resilience against such attacks.

A. Lab setup

Fig. 1 shows a diagram of the lab environment, which consists of a firewall connected to the internet (OS3 institute network) with switches emulating a multi-layered private network with load balancing. For the firewall, we ran our experiments using the popular pfSense firewall distribution, as well as the traditional Linux iptables.

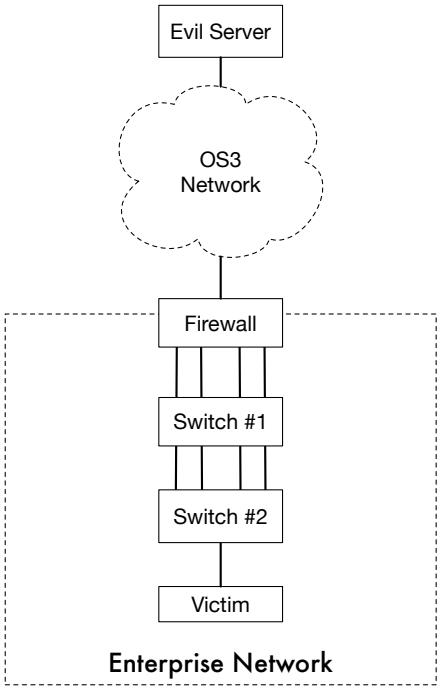

Fig. 1. The lab setup consisting of 4 components: the tracing service, the firewall, switches and victim. The victim contacts the server, while the switches emulate a load-balanced internal network.

The firewall is the entry point of the network: it prevents unauthorized access by segmenting the network in two different zones. The first interface is used to communicate externally to the OS3 network and from there, to the Internet. The second interface is used to communicate to the internal network, which the firewall is supposed to guard against unauthorized access. It is connected to Switch #1 through a trunk with four routed Point-to-Point Virtual LANs (VLANs). Below this, Switch #1 is connected to Switch #2 with four Point-to-Point routed interfaces, running the OSPFv2 (IPv4) and OSPFv3 (IPv6) routing protocol for prefix propagation.

All the network layers operate as routers that perform load-balancing to spread the data over multiple interfaces. The firewall does this by Layer 4 load-balancing, which implies balancing packets by `SRC IP`, `DST IP`, `SRC PORT`, `DST PORT` network and transport layer header fields. The switches, instead, balance only over the `DST IP` header values on the

network layer. This specific constraint is imposed by technical limitations of the *Cisco Catalyst 3750x* device. The reason for implementing load-balancing is to more accurately simulate a corporate network that allows for multiple paths in order to enhance performance by link aggregation or to introduce redundancy.

The x86_64 Linux server acting as firewall was configured to perform routing, via iptables rules targeting the `FORWARDING` chain. We followed the official documentation of the well known Linux distribution Red Hat [12] [13], which recommends the following configuration:

```
1 FORWARD -m state --state ESTABLISHED,RELATED
      -j ACCEPT
2 FORWARD -i eth2 -j ACCEPT
3 FORWARD -o eth2 -j ACCEPT
```

This configuration allows all traffic both from and to the internal interface, `eth2`, or otherwise, only inbound traffic related to a connection initiated from inside (`ESTABLISHED`) or related to previously allowed connections (`RELATED`).

B. Tracing service

The "Evil Server" is a Linux Ubuntu machine that hosts the REVCON web service, which was made publicly available at https://revcon.os3.nl/. The tracing service is running inside an Alpine Linux Docker container, which uses *netfilter* queues to intercept packets once they arrive at the network interface and then decides whether to accept, drop or alter the packet based on the logic we implemented. The service was designed to be used in our internal lab experiments and for our public case study. It can be reconfigured without being restarted and provides access to statistics and the gathered data, via a password-protected admin interface. The scan initiator will be provided with the information obtained during the scan, so that they can observe if their network configuration is vulnerable to this kind of information gathering. The logic for interacting with the *netfilter* queues and tracking the TCP flows is based on the *tracetrout* project [14], which we rewrote in order to provide stable IPv6 support, store experiment results and offer a web based UI for our case study that conforms to the requirements imposed by the European GDPR.

The server has 9 different IPv4 addresses and 8 different IPv6 addresses assigned to its main network interface, in order to provide dual-stack access over both IPv4 and IPv6. This enables us to force path variations through the implemented load balancing, by contacting the server via different addresses. The service is secured with a TLS certificate and the different addresses are resolved via DNS entries to multiple subdomains, following the scheme:

```
ip<numAddr>v<numIPVersion>.revcon.os3.nl
```

C. Tracing procedure

Figure 2 illustrates the probing procedure. To start the parasitic probing, the server provides a REST API via the `/trace` endpoint, that requires a valid traceID and a HTTP parameter to signal explicit user consent. On the client side, the user interface offers two dedicated buttons to start the experiment over either IPv4 or IPv6. Every time a client clicks one of the buttons, multiple traces are initiated via JavaScript by contacting the `/trace` endpoint, passing the mandatory parameters and waiting for the service to close the connection or send back a result.

On the server side, a new connection to the `/trace` endpoint will validate the parameters and, upon successful validation, reply with the HTTP `Connection: Keep-Alive` header to signal that the connection shall not be closed by the client. The service will then write data into the connection and intercept the TCP packets generated by the operating system once they arrive at the network interface to be sent out to the client. The intercepted packet is then modified by setting the TTL value corresponding to the current probing cycle, recalculating the IP header checksum and sending it off to the interface for being delivered to the victim. For the trace endpoint, the server enforces a Server-Sent Events (SSE) session, to allow pushing result data back to the client once available.

In order to prevent an incorrect mapping of a TTL probe result to an ICMP error reply, the sequence number for an ongoing trace to a specific hop is stored by the service and used to identify and map the corresponding reply. This works because the original TCP sequence number is preserved in the first 8 bytes [15] of the TCP header (or within 576 bytes of the total ICMP datagram [16] depending on which RFC the device adheres to), which is being returned in the ICMP error payload.

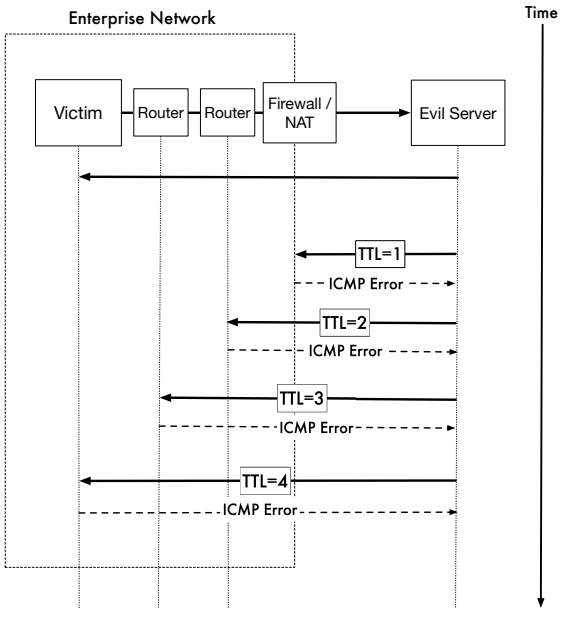

Fig. 2. A simplified overview of the reverse probing procedure and the flow of network packets, ignoring intermediate devices between server and enterprise network.

D. Multi tracing

In order to increase the amount of discovered network nodes and gather as much information about the target network as possible, we've implemented multi tracing with deduplication of the result paths, as shown in fig. 3. Instead of using a single trace connection, a configurable amount of 1-8 connections will be openened simultaneously and, once the last trace has completed, the results will be aggregated and sent back to the client in an easy to read text-based format, while persisting the raw result summary as YAML on the server side. In the deduplication phase, multiple trace summaries will be reduced to a single one, that reflects all observed path variations for each TTL value. This solution reveals whether multiple paths through the target network exist and at which depth in the network they have been encountered.

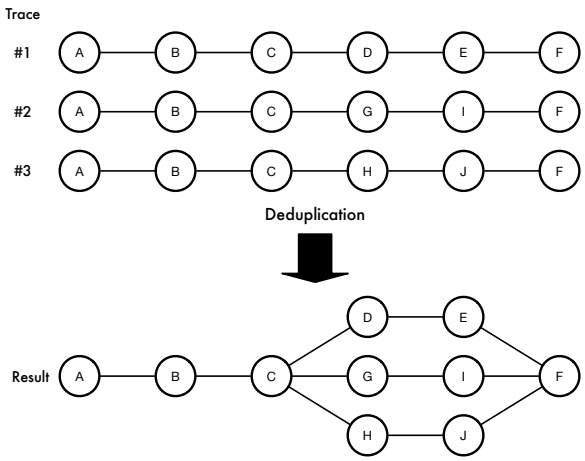

Fig. 3. Multi tracing with path deduplication

E. Case study

The tracing service was built to conform to the Dutch Code of Conduct for Research Integrity [17] and the European General Data Protection Regulation (GDPR) [18]. Every tracing operation was either performed on victim devices under our control or on participants who have agreed to participate in our research project. All recorded data was stored securely, processed confidentially and destroyed after our research. Participants for the case study have been recruited through announcements on public forums such as social media and private contacts of our research institute.

III. Results

During our lab experiments with NATted IPv4 and IPv6 networks, we observed that pfSense would translate the internal source addresses from the ICMP error replies to the public address of the gateway. This continued to occur even when explicitly specifying that packets for internal IPs should not be NATted. Upon further investigation we discovered that this is specified in the NAT RFC 2663 [19], which demands that the payload and the outer IP header must be NATted for all `RELATED` traffic.

We were able to reproduce these results on an Ubuntu 18.04 instance with iptables as a firewall, which also respects the NAT RFC 2663 [19] in that regard. This means that even though the internal hosts are receiving the probes and responding to them, the source of their ICMP error message replies are still being NATted. In practice, this makes it impossible to gather address information about internal hosts behind the network address translation. It is possible however, to observe the depth of the network, since the tracing service will be receiving multiple packets with the same (NAT) source, before stopping the trace because the probe received an `ACK` response, indicating that the destination has been reached.

When executing the traces against the multi-layer enterprise lab network without NAT, it was possible to successfully retrieve private IP addresses through both tested stateful firewalls (pfSense and iptables) for IPv4 and IPv6. As NAT is commonly being deployed for IPv4 networks, it prevents this type of attack in practice. Due to the huge address space for IPv6 networks however, NAT is typically not used and this leaves the network vulnerable, unless the firewall is configured to drop the outgoing ICMP *TTL Expired* replies. However, this is explicitly not advised as per RFC 4890 "Recommendations for Filtering ICMPv6 Messages in Firewalls" [4].

The experiment data we gathered for the evaluation in this report consists of 202 multi traces collected over a period of 7 days. Of these, 150 trace results are IPv4 traces and 52 are IPv6 traces.

Based on those trace results, we conclude the following:

1) IPv4 is in more widespread use than IPv6
2) NAT is used for the majority of IPv4 traces
3) For traces where NAT was used: the depth of the network is visible to an attacker

Additionally, we observed that traces against IPv4 networks where NAT was disabled did not succeed outside the lab environment, because the ICMP replies did not arrive at the tracing service. Since we could observe the ICMP error replies containing private addresses leaving our remote-client network used for testing, we suspect the reason for this being an Internet Service Provider (ISP) along the path is filtering packets with private IP addresses [20].

This assumption was verified by generating packets with a private source IP and forwarding them to a public IP over the internet. These packets were verified to be sent out and not be received by the destination. The ISP in this test scenario was the dutch ISP Ziggo.

Overall the experiments in the case study did not reveal internal IPv4 addresses to us, and no IPv6 addresses from a known private range.

We think this is due to the fact that most traces where executed from users behind their NATted home network and not from our preferred target audience: users inside a corporate network. Another factor is the short experiment duration and our limited reach for recruiting participants.

A. Mitigations

There are two options to protect against this type of reconnaissance as a network administrator:

1) blocking the ICMP *TTL Expired* message from leaving the network
2) blocking the probing packets that trigger the *TTL Expired* message

In order to implement the first option, one could either explicitly block ICMP packets with a source address of the internal addresses passing the firewall or block all ICMPv6 *TTL Expired* packets from leaving the network. Blocking specific private addresses has the drawback that these rules need to be updated every time the network architecture is modified, for example when new hosts are added. This introduces the possibility to miss a newly added device and therefore allows its address to leak, aside from the administrative burden. Blocking all ICMPv6 *TTL Expired* packets from leaving the internal network on the perimeter firewall solves this problem, but makes the network harder to debug for legitimate troubleshooting. This might lead to an administrator removing the restriction for such purposes and forgetting to put it back in after troubleshooting is done.

The following shows an example of an iptables rule to `DROP` all responses from an internal IP:

```
1 Chain FORWARD (policy DROP)
2 target   prot  in   out   source      destination
3 DROP     all   *    *     <IP to hide>  0.0.0.0/0
4 ACCEPT   all   WAN  LAN   0.0.0.0/0     0.0.0.0/0
        state RELATED,ESTABLISHED
5 ACCEPT   all   LAN  WAN   0.0.0.0/0     0.0.0.0/0
```

Due to the described complications of the first solution, we recommend the second approach. For this, a rule has to be applied that only allows packets from outside to inside to pass through, if they have a TTL that is guaranteed to arrive at its destination unexpired.

This requires a network administrator to keep track of his network depth, but we argue that this will change less often than the number of devices in the private IP address range. An example of this would be blocking all packets with a $TTL < 10$. This would mitigate the discussed attack for networks with a depth of 0 to 9 hops.

From our testing we observed a host from New Zealand being routed through Japan take 18 hops (18 TTL's being decremented) before arriving at our service. Disallowing all $TTL < 10$ would not be a problem in this case, as Linux defaults to a TTL of 64, while Windows defaults to a TTL of 255 for sending packets. The average path length through the internet is 5-6 hops [21].

An example of request `DROP` ruleset below:

```
1 Chain FORWARD (policy DROP)
2 target   prot  in   out   source      destination
3 DROP     all   WAN  *     0.0.0.0/0     0.0.0.0/0
        TTL match TTL < 10
4 ACCEPT   all   WAN  LAN   0.0.0.0/0     0.0.0.0/0
        state RELATED,ESTABLISHED
5 ACCEPT   all   LAN  WAN   0.0.0.0/0     0.0.0.0/0
```

It is important to keep in mind that if the firewall requires the external Border Gateway Protocol (eBGP) to run on the outside interface, TCP port 179 must be allowed to be accessed from the specific peer with a TTL of 1.

IV. Conclusion

We discovered that, when using IPv6, the reverse traceroute technique is more likely to successfully gather information due to the following reasons:

1) NAT is not usually in place for IPv6 as its use is discouraged [22] [23]
2) ICMPv6 packets are usually not blocked by networks, as proposed by RFC 4890 [4], paragraph 4.3.1:

   Time Exceeded (Type 3) - Code 0 must not be dropped.
3) Internal addresses are easily identified in IPv4, as they consist of well known private addresses. Because it is advised to use global addressable IPv6 [24] [25] addresses, the firewall will have to maintain a list of all privately used public IPv6 addresses, in order to prevent them from leaking
4) In our internal testing we observed an ISP blocking all packets from the private IPv4 address range (RFC 1918) [20] addresses. For IPv6 this will be infeasible (see 3)

Furthermore, due to the load-balancing we implemented in our lab, we were able to prove that differentiating the source IP addresses of our tracing service, leads to the detection of multiple paths in the lab-network. Our results from participants, that tested on live networks by using our web service, confirm that we are able to detect multiple paths in a real-world scenario.

We conclude that this type of reconnaissance is feasible against IPv6 networks, due to the absence of NAT and the existence of technical constraints that allow the probe responses to leave the target network. For IPv4 networks this attack seems to be almost impossible, since NAT is almost always enabled, due to address space limitation. Detecting multiple paths in load balanced environments is possible and can be used to increase the number of discovered hosts in the target network. We've also discussed possible mitigations, as well as their implications and drawbacks.

A. Discussion

This type of parasitic reconnaissance would be trivial to implement for a cloud provider, enabling internal network graphing at scale. When consumers utilize cloud servers, the provider could be injecting packets with low TTL values into the connection, not impacting the clients speed and usability. A cloud provider, or any cloud service, could thereby map public and private networks. As a client authenticated himself to the cloud, the cloud provider already knows the identity of the user. If the cloud provider also knows the internal intermediate nodes, it could know when this specific client has moved to a target area. Because the TTL field exists in the IP layer, any other layer above the aforementioned could be used to perform

this type of reconnaissance. This would include all types of common network and internet communications, including:

- transport layer protocols: TCP, UDP, ICMP
- application layer protocols: HTTP(S), SSH, SIP, DNS

A consideration to take into account is the possibility of a victim being behind a VPN. In that case, the gathered trace information could help to identify if the VPN service is hosted company internal or hosted by an external provider. The first case is especially interesting for an attacker, since it could potentially reveal information about the company internally hosted VPN service. This knowledge could be used for penetrating it and thus being able to manipulate a central security mechanism of the corporate network.

We implemented the tracing service in a way it enables network administrators to test their internal networks and defenses, making sure they aren't leaking private information due to a misconfiguration of their network setup. Currently, each probing packet to a hop will generate a fresh packet and therefore increment the TCP sequence number, the TCP connection will be impacted once the victim finally receives the probe, because the TCP implementation notices a gap in sequence numbers caused by all probes that have timed out previously. This will lead to the client requesting the missing sequence number from the service, thus looking like packet loss to an observer. Although this could be addressed by reusing the same packet for all probes towards the same hop, we deliberately chose the first approach, as we expect stateful firewalls to drop multiple packets arriving with the same sequence number.

V. Future work

A novel strategy for REVCON could be employing it on a large scale instead of the discussed targeted attack scenario. This could be accomplished by, e.g., embedding malicious JavaScript code, used for triggering a trace without user consent, into advertisements and thus achieve its execution on a large number of users at a global scale. Adding support for UDP as a transport protocol would allow extending the number of possible applications. We imagine REVCON to be useful as a tool for penetration testing and will extend it to conduct the tracing without user consent, masquerading an ongoing trace as a redirect. The implemented SMTP integration could be used to generate phishing mails in an automated manner, in order to send mails with a link to the tracing service to the victim.

The REVCON project will be released under the GPLv3 license, to allow other researchers to experiment with this mechanism for the purpose of information gathering. Additionally, we are considering to host the service permanently with disabled data storage, to provide network administrators the possibility of a quick self check. The repository will be made available at https://github.com/dreadl0ck/revcon.

References

[1] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[2] R. Hinden. Ip version 6 addressing architecture, 2006.

[3] Should i block icmp? http://shouldiblockicmp.com, 2017.

[4] Icmpv6 filtering recommendations, rfc 4890, 2007.

[5] David Goldsmith and Michael Schiffman. A traceroute-like analysis of ip packet responses to determine gateway access control lists. *Outubro de*, 1998.

[6] Van Jacobson. *traceroute(8) Linux User's Manual*. The Linux man-pages project, https://www.kernel.org/doc/man-pages/, 2.1.0 edition, October 2006.

[7] Kang Leng Chiew, Kelvin Sheng Chek Yong, and Choon Lin Tan. A survey of phishing attacks: their types, vectors and technical approaches. *Expert Systems with Applications*, 106:1–20, 2018.

[8] Paratrace - sneaky tracerouting.

[9] 0trace, 2007.

[10] Ivan Morandi, Francesco Bronzino, Renata Teixeira, and Srikanth Sundaresan. *Service Traceroute: Tracing Paths of Application Flows: Methods and Protocols*, pages 116–128. Springer, 03 2019.

[11] Rob Sherwood and Neil Spring. Touring the internet in a tcp sidecar. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 339–344, New York, NY, USA, 2006. ACM.

[12] Redhat guide for forward and nat rules, 2019.

[13] Redhat guide for iptables and connection tracking, 2019.

[14] HowNetWorks. tracetrout, 2018.

[15] Internet control message protocol, rfc 792, 1981.

[16] Requirements for ip version 4 routers, 1995.

[17] VSNU. Netherlands code of conduct for research integrity, 2018.

[18] European Commission. General data protection regulation, 2019.

[19] Ip network address translator (nat) terminology and considerations, 1999.

[20] Address allocation for private internets, 1996.

[21] Shiva Kasiviswanathan, Stephan Eidenbenz, and Guanhua Yan. Geography-based structural analysis of the internet. *Los Alamos National Laboratory*, 1 2010.

[22] Local network protection for ipv6, 2007.

[23] Iab thoughts on ipv6 network address translation, 2010.

[24] Depricating site local addresses, 2004.

[25] Considerations for using unique local addresses, 2017.