Practical Network Defense

Fourth Assignment - University "La Sapienza"

Group 27: Nicola Bartoloni 1909908 - Valerio Trenta 1856471

2020-23-06

1 Initial comments and scope

This last assignment is meant to provide an additional level of defense to the **ACME CO. network** by setting up **Intrusion Prevention Systems**, both hosts-based and network-based, in the two firewalls of the network and in the hosts of the **DMZ** subnetwork.
Once these systems are fully operational, a **vulnerability assessment** is then carried out to find, examine and pinpoint the vulnerabilities that are still affecting the services that the ntetwork provides.
Given the goal of the **NIPS** that we are going to setup in the two firewalls - that is, to detect and prevent intrusions in the whole network, especially the one set up at the **Main firewall** - and given the fact that we have to consider all the services that the target network provides, the scope of the assignment is the entire network.
Please keep also in mind that the vulnerability assessment has actually its own scope and scenario, which is described in the fourth chapter.

2 Intrusion Prevention Systems setup

It is explicitly required in the assignment to actually set up **Intrusion Prevention Systems**, so that they can not only detect intruders raising an alarm, but also perform actions in order to prevent the intrusion, like dropping suspicious packets. We are going to set up two different systems: in the two firewalls, we exploit the **suricata** service provided by the **OPNSense** administration panel to build a **Network Intrusion Prevention System**, while in the two hosts of the **DMZ subnetwork**, we download and configure the **fail2ban** tool to build a **Host Intrusion Prevention System**.
The differences and analogies between the two systems will be clear in the next two sub-paragraphs, as well as their configurations and functionalities.

2.1 suricata

The **suricata** service can be enabled in the **OPNSense** administration panel by navigating to *System -> Services -> Intrusion Detection -> Administration*. For both routers,

we enable the service and we also check the box **IPS** to make it possible for the service to actually *drop packets* matching a rule, so that it will not only raise an alarm, but also perform some concrete action in preventing the intrusion.
For both of them we also enable the **promiscuous mode** so that the systems will also monitor and check the rules for packets that are not strictly directed to one of their interfaces - so, obviously, forward packets will also be checked - and the **logging** service.
In the **Main Router**, we enable the service for *all the interfaces, but the WAN interface*. This latter interface indeed is probably the one seeing the most of the traffic, being connected to the Internet, so we decide to run the system only at the interfaces directly connected to our services, so that when packets get there they have already been checked against the WAN interface's firewall rules and the interface itself won't be a bottleneck to the whole network just by checking every packet to every single rule of the **NIPS**.
In the **Internal Router**, the very same logic is applied and thus the service is only enabled on the **CLIENTS** and **SERVERS** interfaces.
Both systems in both firewalls are configured to update their rules daily according to a given schedule. Both of them will drop packets matching against one of the rules that have been selected.
The rulesets that are enabled follow below - and they are almost the same for both to apply defense-in-depth for the most internal firewall:

- all the rulesets from **Abuse.ch**, except *Feodo Tracker*. These rules should detect compromised websites distributing malware, compromised and bad SSL certificates and such;
- **ETOpen Attack-Response** catching common results of successful attacks;
- **ETOpen botcc** for detecting Botnets and C2 hosts;
- **ETOpen DOS** rules for DoS attacks;
- **ETOpen DROP** rules for spam activities;
- **ETOpen Exploit** and **ETOpen Malware** for detecting general exploits and malwares;
- **ETOpen Inappropriate** rules for detecting accesses to inappropriate websites;
- **ETOpen Policy** rules containing checks for things that are usually not enabled by a company policy - that seemed our case;
- **ETOpen Scan** rules to prevent network scans, probes and in general reconnaissance activities;
- **ETOpen Web rules** - *client* in the Internal firewall, *client and server* at the Main firewall, for web application security and prevention of common attacks - such as SQL injection, XSS;

- finally, we also included some rules from the **OPNSense Suricata Application Detection**, namely for **file−transfer** and **mailing** applications - once again, it seemed our case to include these rulesets given the network belongs to a company, and clients are likely to be exploiting such applications.

The two routers do not seem to be overloaded by the rulesets that were applied, even if they are quite a lot and some might also be unnecessary - **botcc** and maybe the **Inappropriate** rules, the latter especially given the proxy and DNS service we set up previously.

2.2 fail2ban

The **fail2ban** tool can be downloaded on both the two machines running in the **DMZ**, namely the **proxy machine** and the **web machine**. We have three different services running on these machines, whose access we want to protect with this tool:

- **SSH** service running on both;
- **apache2** service running on the **web machine**;
- **squid** service running on the **proxy machine**;

so we copy the content of file */etc/fail2ban/jail.conf* in a new file, *jail.local* under the same directory, which will be now used by **fail2ban** as configuration file for its jail. We leave unmodified the preconfigured values for parameters such as *bantime* and *maxretry* - 3600 and 5 - and we do not specify in neither of the two configurations for the two machines IP addresses that we want to ignore in the field *ignoreip*, apart from the local loop, so that especially in the case of the **squid** service, we will monitor also accesses coming from the Clients subnetwork, so to prevent possible physical intrusions on the machines of the aforementioned subnet which may then lead to a misuse of the proxy service. The tool will then monitor the log files of the *enabled* services in the *jail.local* file to detect possible violations of authentication on those services - e.g., brute force attacks - and **ban** the hosts which made a number of attempts to authenticate that is $>$ *maxretry*. To each services that we wish to monitor through the tool, we add the string *enabled = true*. These services are:

- **sshd** in both the machines;
- **apache2** and related lines in the *jail.local* file for the **web machine**;
- **squid** for the **proxy machine**.

We enable and restart the **fail2ban service** through command line *service fail2ban restart*. To check the list of jailed hosts per each service, we can execute command *fail2ban-client status <service>*, as pictured in the figures below.

Please keep also in mind that we know that at least one more process is running on both machines - the **rsyslog** process for remote logging at log machine in the Internal

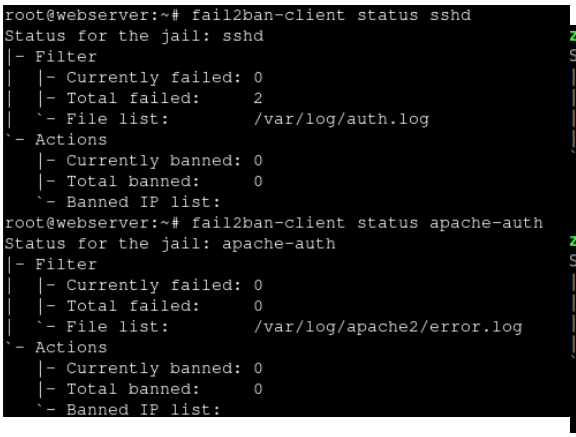


Figure 1: fail2ban jails for services in the Web machine (apache-auth is just one of the several apache jails we enabled).

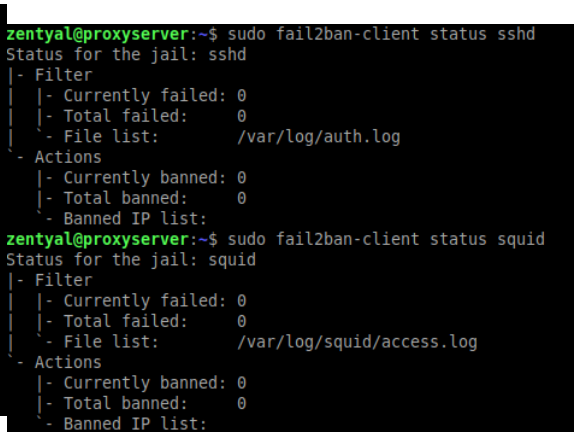


Figure 2: fail2ban jails for services in the Proxy machine.

servers - but this does not require any authentication mechanism so it is not monitored by the tool.

3 Engaging the Assessment

In this section we are going to define the **scope** of the assessment, as weel as the typology of assessment we are going to carry out - i.e., which subnetworks we are going to test, and how prepared these subnetworks should be. Since we want to base our test upon the voices we heard in the company, if there is a new platform that is being developed, it is very likely to be placed in the **DMZ subnet**: indeed, it is likely that a platform belonging to our company is going to accept connections from visitors or external partners, and the **DMZ** is where this should happen.
Furthermore, our platform is surely going to exploit the services provided by our **Internal servers** - i.e. the logging service and the DNS service. This is why we should focus our assessment on these two subnetworks, so **DMZ** and **Internal servers** are going to be the subnetworks which define the perimeter of our assessment, our scope.
While the operations of **Linux hardening** we performed previously should not be rolled back during this assessment, we should ask ourselves how the other security measures we implemented are going to affect the tests als with respect of the typology of assessment we want to make. In order to gather as much information as possible about these two subnetworks, indeed, we should carry out the assessment with almost every security measure in the targets disabled. This is also due to the fact that the **GSM** machine is actually inside one of our target subnetworks (**Internal servers**), and while for the hosts in this subnet security measures such as firewalls or IPS are not going to affect the scan - traffic between scanning machine and targets is not going through any firewall at all - these measures could actually affect the scanning procedure of hosts in the **DMZ** subnet: indeed, we tried and found out that, with the firewall rules enabled, **GSM** is not able to perform the scan. This means that, to actually be able to find vulnerabilities in that subnetwork, we should at least get rid of the firewall - as if the **GSM** machine was in the same subnet. Obviously it is not, so the vulnerabilities that we are going to find out, have to be considered such in the worst case scenario - that is, an adversary who gained access to these two subnetworks and can send/receive Ethernet packets in there, for example gaining phyisical access to them or by performing some kind of spoofing attack.
The assessment results that we are going to describe and evaluate in the next section are, thus, obtained on the target hosts *with firewalls and IPS (fail2ban and suricata) disabled.*

4 Vulnerability Assessment

We are going to provide the results for each host on which the assessment was performed through **openVAS**, divided by subnet. We are going to adopt the **Severity Score** provided by **openVAS** for each of the vulnerabilities - on a scale from 0 to 10, each vulnerability is assigned a value that ranges in **None, Low, Medium, High** to describe its risk level. These values, along with their **Quality of Detection** score and the context, are then exploited to prioritize the most severe vulnerabilities in the **Analysis** subsection,where the vulnerabilities identified are discussed and mitigations are provided.
Every assessment was carried out requiring a minimum **QoD** of 70%: we need to keep this in mind, since this minimum value, when met, means that the remote detection performed is not fully reliable.
Also, please notice that we are omitting several other information that **openVas** classified with severity score **Log** - 0.0. The only purpose of these information is to point out the results of different tests carried out by the tool that do not highlight vulnerabities by themselves, but are useful in the whole process and to us, as analyst, to check how the *discovery, port scan, service detection and OS detection* phases were carried out by **openVAS**: some examples of these activities may be the execution of the **traceroute** command, **OS fingerprinting** operations, the **Services** log which points out all the services that were detected on the machine, **SSH** detection with corresponding algorithms to perform **encryption** and scans to determine which **cryptographic protocols** are exploited on the machine to perform **SSL/TLS**.
For each machine, the vulnerabilities to prioritize are highlighted in **bold**.

4.1 Results

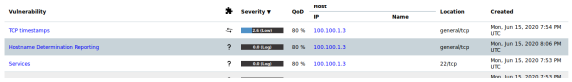
4.1.1 Internal Servers

DC machine - 100.100.1.2

- **SSL/TLS Missing Secure Cookie Attribute - Severity Score: Medium**;
- **Missing httpOnly Cookie Attribute - Severity Score: Medium**;
- **SSL/TLS Certificate Signed Using A Weak Signature Algorithm - Severity Score: Medium**;
- TCP Timestamps - Severity Score: Low;

Figure 3: Vulnerabilities detected in the DC machine.

Log Server Machine - 100.100.1.3



Vulnerability	Severity ▼	QoD	Host IP	Host Name	Location	Created
TCP timestamps	2.6 (Low)	80 %	100.100.1.3		general/tcp	Mon, Jun 15, 2020 7:54 PM UTC
Hostname Determination Reporting	0.0 (Log)	80 %	100.100.1.3		general/tcp	Mon, Jun 15, 2020 8:06 PM UTC
Services	0.0 (Log)	80 %	100.100.1.3		22/tcp	Mon, Jun 15, 2020 7:53 PM UTC

Figure 4: Vulnerabilities detected in the Log Server machine.

- TCP Timestamps - Severity Score: Low;

GSM machine - 100.100.1.4

- SSL/TLS: Untrusted Certificate Authorities - Severity Score: Medium;

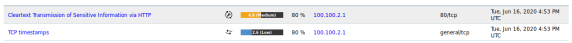
Vulnerability	Severity ▼	QoD	Host IP	Host Name	Location	Created
SSL/TLS: Untrusted Certificate Authorities	5.9 (Medium)	99 %	100.100.1.4		443/tcp	Mon, Jun 15, 2020 8:04 PM UTC
SSL/TLS: Report Perfect Forward Secrecy (PFS) Cipher Suites	0.0 (Log)	98 %	100.100.1.4		443/tcp	Mon, Jun 15, 2020 8:03 PM UTC
Response Time / No 404 Error Code Check	0.0 (Log)	80 %	100.100.1.4		80/tcp	Mon, Jun 15, 2020 8:02 PM UTC

Figure 5: Vulnerabilities detected in the GSM machine.

4.1.2 Firewalls

Internal Firewall

- **Cleartext Transmission of Sensitive Information via HTTP - Severity Score: Medium**;
- TCP timestamps - Severity Score: Low;



Cleartext Transmission of Sensitive Information via HTTP	4.8 (Medium)	80 %	100.100.2.1		80/tcp	Tue, Jun 16, 2020 4:53 PM UTC
TCP timestamps	2.6 (Low)	80 %	100.100.2.1		general/tcp	Tue, Jun 16, 2020 4:53 PM UTC

Figure 6: Vulnerabilities detected in the Internal Firewall machine.

Main Firewall

- **Cleartext Transmission of Sensitive Information via HTTP - Severity Score: Medium**;
- TCP timestamps - Severity Score: Low;

Cleartext Transmission of Sensitive Information via HTTP	4.8 (Medium)	80 %	100.100.6.1		80/tcp	Tue, Jun 16, 2020 5:02 PM UTC
TCP timestamps	2.6 (Low)	80 %	100.100.6.1		general/tcp	Tue, Jun 16, 2020 5:02 PM UTC

Figure 7: Vulnerabilities detected in the Main Firewall machine.

Figure 8: Vulnerabilities detected in the Web Server machine.

4.1.3 DMZ

Web Server Machine - 100.100.6.2

- TCP timestamps - Severity Score: Low;

Proxy Machine - 100.100.6.3

- **SSL/TLS Missing Secure Cookie Attribute - Severity Score: Medium**;
- **Missing httpOnly Cookie Attribute - Severity Score: Medium**;
- **SSL/TLS Certificate Signed Using A Weak Signature Algorithm - Severity Score: Medium**;
- TCP Timestamps - Severity Score: Low;

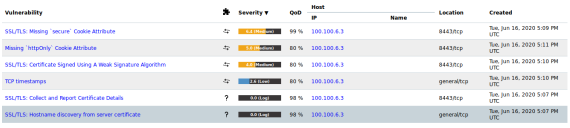


Figure 9: Vulnerabilities detected in the Proxy machine.

4.2 Analysis

Obviously, the fact that machines implementing the same services - **DC** and **Proxy** with **Zentyal**, both **firewalls** with **OPNSense** - were also diagnosed with the same vulnerabilities, is not simply due to chance. This is an advantage, since when we know how to fix a vulnerability in one of those machines, we also know how to fix it in the other one. We are thus now analyzing one by one the prioritized vulnerabilities - aslo explaining *why* we chose to prioritize them - and then those that are left.

4.2.1 Cleartext Transmission of Sensitive Information via HTTP

- Severity Score: **Medium**;
- Affected Machines: Internal Firewall, Main Firewall (OPNSense);

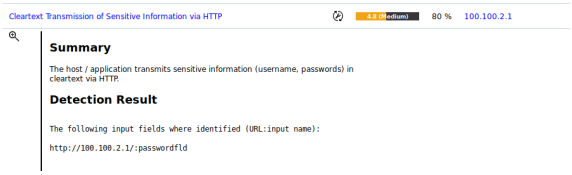


Figure 10: Password is sent in cleartext through HTTP.

One of the desired features of a Firewall, is that of being immune to penetration. Firewalls are the first devices we should protect in our network, as they filter the traffic that is destined in/out from our subnetworks, and they also implement several other services (DHCP, suricata, OpenVPN...). Not to mention their importance in routing. This vulnerability affects the **OPNSense** service, and is due to the fact that **HTTPS** is not enabled.
This is probably the most important vulnerability to mitigate due to its potentially huge negative impact on our network and to the relatively easy fix.
Suggested way to mitigate this is indeed by checking at both firewalls the **HTTPS** box on option **Protocol** under $System -> Settings -> Administration$ as pictured in **Figure 11**: this way, authentication parameters - as well as any other data - should be sent through an encrypted channel. Since an adversary sniffing the devices' traffic might have already exploited this vulnerability, once mitigated it is also recommended to at least change all the passwords for each user in both firewalls.

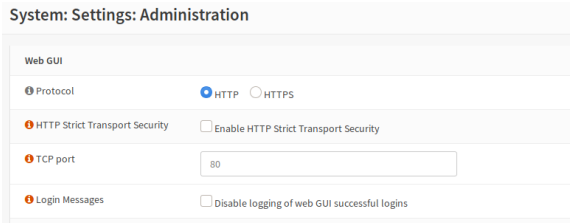


Figure 11: Suggested fix.

4.2.2 SSL/TLS Certificate Signed Using A Weak Signature Algorithm

- Severity Score: **Medium**;
- Affected Machines: DC, Proxy (Zentyal);

The certificate provided for SSL/TLS sessions by both the machines is signed through **SHA-1 algorithm**, which is publicly known to be, nowadays, **not secure**, due to the fact that in 2017 the **Google Security Team** announced to have broken the algorithm by finding a collision. This means the certificate can be forged, it is possible to impersonate both machines and a *MITM* attack is technically feasible on them. To mitigate this threat, it is recommended to generate a new certificate, one for each service running in the two

Figure 12: Certificate is signed with a weak algorithm - SHA1.

machines, signed with a different algorithm, e.g. **SHA-250** or **SHA-3**. We do not know, though, whether this fix can be applied directly by us or it requires the vendor of the product.

4.2.3 SSL/TLS Missing Secure Cookie Attribute

- Severity Score: **Medium**;
- Affected Machines: DC, Proxy (Zentyal);

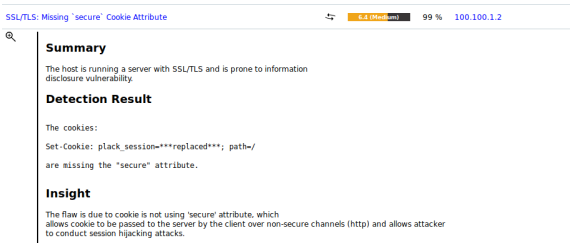


Figure 13: Cookies could be sent through an unencrypted channel.

Zentyal service, reachable at port **8443** in both machines, already exploits the **HTTPS** protocol, so the server is running with **SSL/TLS**, but the *Secure* flag is not sent for the cookies that may be used to remember an already authenticated user - meaning the cookies could be sent in clear text. This would allow an adversary which sniffs the traffic to read the cookie an authenticated client sends to the server, if the client sends it through an unencrypted channel, and exploit it to *hijack his session* and be authenticated by **Zentyal** as an user with privileges, potentially leading to a disruption of the services running in the machines. We found no ways to actually instruct **Zentyal** on which cookies to accept and with which parameters - this might be a feature only available in the commercial edition, or only to the vendor of the product itself.
Beacuse of this, our only way to mitigate this vulnerability may be to make sure that no adversary is able to join the subnetworks of the two **Zentyal** machines and perform eavesdropping.

4.2.4 Missing httpOnly Cookie Attribute

- Severity Score: **Medium**;

- Affected Machines: DC, Proxy (Zentyal);

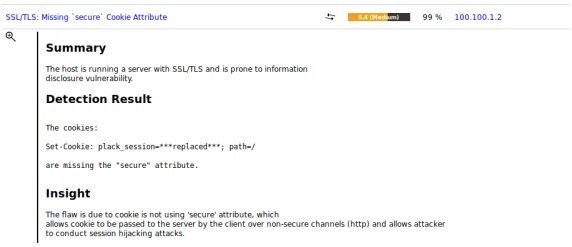


Figure 14: Cookies can be accessed through JavaScript code.

This vulnerability is similar to the previous one, except it describes the cookies set by **Zentyal** to be missing the **httpOnly** attribute. This flag in a cookie prevents it from being accessed by scripts at client side - e.g. a JavaScript script accessing the cookie through the DOM of the web page. Just like the previous vulnerability, this might lead to *session hijacking* as a result of *XSS attacks*.
The only way to mitigate this vulnerability is the one we already highlighted in the previous subsection about **SSL/TLS Missing Secure Cookie Attribute** vulnerability.

4.2.5 Low priority vulnerabilities

1. SSL/TLS: Untrusted Certificate Authorities

- Severity Score: **Medium**;
- Affected Machines: GSM;

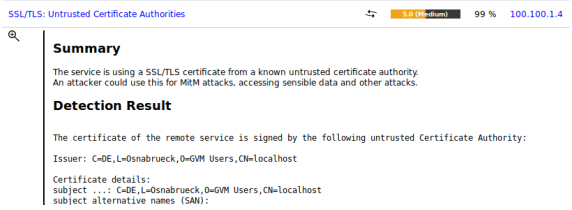


Figure 15: GSM - untrusted CA.

This is probably a minor, false alarm: the CA signing the certificate for the *GSM* machine is **Greenbone** itself, the scanner doesn't recognize it as a public, trustworthy CA, but we as users know it can be trusted.

2. TCP timestamps

- Severity Score: **Low**;
- Affected Machines: all, except GSM machine;

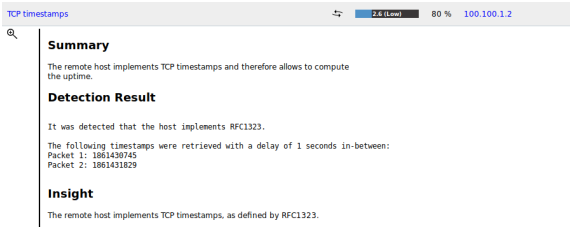


Figure 16: TCP Timestamps vulnerability in DC machine.

TCP timestamps allow to compute the uptime of each host in the network. This provides extra information to an attacker, such as knowing when was the last time the machine was rebooted - which is something that occurs likely, for example, to install patches and updates. This might lead adversary to guess whether the system has installed the latest patches and updates, which might provide protection against latest exploits found on the system, or not, as well as to other exploits. However, hiding **TCP timestamps** for these reasons is basically *security through obscurity*, which is commonly known to be a bad practice: real protection should be provided by always checking periodically the last updates and patches, and install them.
If needed, mitigation is provided by **openVAS** itself in the picture below, and can be easily applied to all the machines - but this might need further analysis, since disabling the TCP timestamps may also affect the system's performances as well as disable some other security features (see next section).

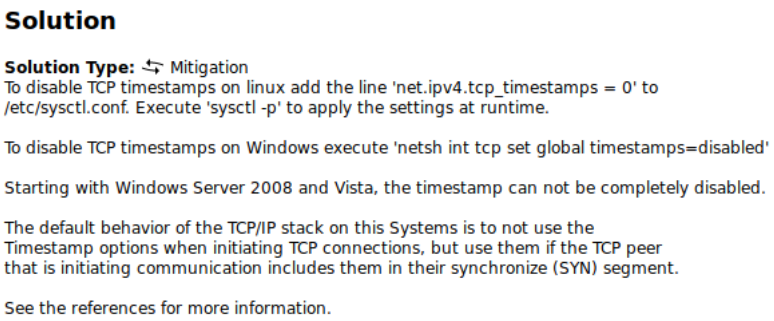


Figure 17: TCP Timestamps mitigation.

5 Vulnerability Assessment: Final Remarks

We highlighted the vulnerabilities that we deemed to be more dangerous - i.e., the ones that need to be prioritized according to our evaluation. Nonetheless, please notice that we were able to detect them because some of the services - e.g. firewalls and IPS - were disabled.
Furthermore, some of the prioritized vulnerabilities affecting the machines running the **Zentyal** service - *secure and httpOnly* flags set for the cookies - might not have an easy fix or mitigation available without either upgrading the service to the commercial version or updating the service itself so that the vendor help is required, meaning that there may be no other choice than to accept them. Because of this, it might be a good idea to deal with the **TCP Timestamps** vulnerability instead, because a well-known mitigation is available and already provided by the **openVAS** tool, and even if its **Severity Score** is actually **Low**, its risk level may be actually higher since it basically affects all the machines in the two target subnetworks and firewalls.
However, some testing should be performed first to actually evaluate whether these timestamps can actually be exploited to guess the uptime of the machines, then in order to evaluate the performance after disabling the timestamps, and again check whether by disabling them no further vulnerabilities are introduced in the hosts. To this end, we provide some other articles as references which migh be starting points to deepen the problem.

- https://www.whonix.org/wiki/Disable_TCP_and_ICMP_Timestamps
- https://cloudshark.io/articles/tcp-timestamp-option/
- https://hackinparis.com/data/slides/2015/veit_hailperin_still_exploiting_tcp_timestamps.pdf