Contents

[Abstract 2](#_Toc120913588)

[1. Introduction 2](#_Toc120913589)

[1.1 Problem Statement 2](#_Toc120913590)

[1.2 Solution Statement 3](#_Toc120913591)

[1.3 Report Structure 3](#_Toc120913592)

[2. Proposal 3](#_Toc120913594)

[3. Related Works 4](#_Toc120913595)

[4. Methodology 4](#_Toc120913596)

[4.1 Network Building Model 4](#_Toc120913597)

[4.2 Net-Scan Discovery Model 4](#_Toc120913598)

[4.2.1 Background 5](#_Toc120913599)

[4.2.2 Probing the Network 5](#_Toc120913600)

[4.2.3 Scanner Tuning 5](#_Toc120913601)

[4.2.4 Sampling Method 6](#_Toc120913602)

[4.2 .5 Acquiring Metrics 6](#_Toc120913603)

[5 Evaluation 7](#_Toc120913604)

[5.1 Performance 7](#_Toc120913605)

[6. Limitations 12](#_Toc120913606)

[7. Future Work 12](#_Toc120913607)

[8. Conclusion 12](#_Toc120913608)

[Acknowledgment 12](#_Toc120913609)

[References 13](#_Toc120913610)

[Appendix A 14](#_Toc120913611)

Network Access Confirmation & Segmentation Testing

Joshua Smith

Master of Science in Cybersecurity

College of Computing, Georgia Institute of Technology

Atlanta, GA

Jsmith701@gatech.edu

# Abstract

This paper aims to solve the problem of manual network access confirmation and segmentation testing related to the PCI compliance mandate. The project created a better, standardized approach to segmentation and access testing. The project goal was to reduce scan time requirements for the scanning, analysis, and reporting. The deliverable provides an out-of-the-box solution that automatically tests access and segmentation of the networks scoped to meet or exceed PCI mandate with less overhead in time and labor.

# 1. Introduction

This section outlines the purpose of this report by outlining the problem, solution, and structure of the remaining information.

# Problem Statement

Much of the penetration testing market is driven by compliance mandates. One of the excellent compliance standards is the Payment Card Industry Data Security Standard which requires penetration testing annually (PCI). Without compliance, a company can be barred from processing transactions on payment processing platforms such as American Express, VISA, and others (Woock). Without payment processing in place, most businesses will incur severe losses in operational revenue alone. In less extreme circumstances, the company will incur fines for non-compliance ranging from $5,000 to $10,000 per month (Baykara). Kicking off pen testing engagements with PCI DSS-driven customers requires attention. Often, customer organizations do not have an inventory management system to track all the servers in their PCI DSS cardholder data environment, never mind any documentation on adjacent networks for which segmentation is required. Testers are forced to scan the entire scoped subnets without any inventory management intel. There’s also no way for the customer to reasonably provide a list of currently allocated IP addresses without some inventory management system. When the customer provides one or more Class A networks, we are talking about a minimum of 16,777,216 potential live hosts and a week or more of scan time (IP Address Class). The span of time needed to check every single address is substantial. Even just running ping sweeps on such vast ranges takes extended periods, and often customer networks do not have ICMP enabled. When ICMP is disabled, the tester is forced to run an array of port scans, cross-examine scan results, and then compare the results to the given scope subnets to determine active host counts in each subnet issued. If active hosts are found to be reachable within a given subnet, then it is confirmed that access to that subnet has been, in fact, granted.

In addition, segmentation testing is required by PCI mandate on an annual basis (PCI). A tester must go through the same tedious process of miscellaneous ping sweeps, port scans, cross-examination of scan results, and comparison to the given subnet ranges to determine if segmentation is in place preventing connections from being built to and from the cardholder data environment.

Returning to the topic of access testing, each time a scoped subnet is not reachable, the tester must wait for the customer network changes to be made and then retest. This process can occur multiple times before access is appropriately placed. Given the time sink of this process and then considering that it can require numerous cycles before a customer can get correct network configurations in place, this for testers results in a very time-consuming and grueling campaign. This project aims to reduce the scan runtime and labor requirements as well as eliminate the analysis and reporting of both access testing and segmentation testing.

Currently, there is no detailed standardized way of testing segmentation and access (PCI). Testers must manually setup and run ping sweeps and port scans of various kinds to cross-check the results only then with other scans along with the given subnets of interest to decide, “Do I have the access expected so that I can accurately test all of the PCI infrastructures?” and “Does segmentation, in fact, exist between the cardholder data environment and adjacent networks?”

# Solution Statement

This project aimed to simplify the verification and proof process for access and segmentation testing for PCI DSS scoped IP subnet ranges. The solution goal was to increase the efficacy of segmentation testing for PCI compliance and confirm access to the required CDE and adjacent environments per PCI mandate. Currently, the compliance standard does not explicitly dictate what qualifies as a ‘good enough’ segmentation test. My solution provides a more comprehensive scanning approach, increases efficacy, sets standardization for segmentation and access testing, and produces proofs that will virtually eliminate human labor to complete.

# Report Structure

The layout of the remaining sections of the report will go over the details of the devised solution. Section 2 will introduce the solution’s approach; Section 3 will cover others’ prior attempts to solve the segmentation testing dilemma. Section 4 will go through the details of the solution, its implementation, and the outcomes, along with supporting metrics. Section 5 will go over the data collected and evaluate the performance of the tooled solution. Section 6 will cover the limitations of the solution and the limits of testing the tool, followed by Section 7, which covers future work for improvement. Then finally, in Section 8, the conclusion, acknowledgments, and references will be announced.



# 2. Proposal

The proposal is to leverage Python to implement an array of ping sweep and port scan techniques to check access and segmentation in an automated fashion. Results will then be compared across different scans derived from various tools, including Fping, Nmap, and UnicornScan, to account for reachable hosts in each assigned subnet range. Once accounting has been completed, the results will be output to a CSV file, organized by the number of live hosts in each subnet, as evidence for management to view in Excel. The Python-driven tool will automate the scanning, accounting, and reporting processes. My approach will be the first to meet all aspects of PCI segmentation testing requirements and access verification for all pen test engagements. In addition, Binderscan could become the standard for the segmentation testing process where no detailed standard currently exists (PCI).

# 3. Related Works

A thorough search of the internet yielded only one tool similar to Binderscan. The tool, “El Segmentador,” written by Yozer Esneider Garcia Marulanda, can be seen publicly on GitHub (Marulanda). This tool resembles Binderscan; however, it is more simplistic as it does no automatic analysis, scan optimization, or any sampling technique to reduce run time. This related work is written in bash. While the tool contains ping sweeps, TCP, and UDP scans, it lacks fine-tuning of scan parameters, applies no intelligent decision-making, and delivers no analysis and reporting capabilities. The lack of strategic control flows, analysis, optimization, and reporting leaves much to be desired. Also, this tool leaves the user with a manual review of disparate reports for cross-examination, all shoved in different directories. In addition, this related work doesn’t account for access testing at the beginning of an engagement and thus has less utility.

# 4. Methodology

My solution is to first to build randomized networking environments varying in size (Class A through Class C) with a percentage of IP space filled with live machines running various operating systems and supporting multiple services. Once these environments are instantiated, the customized scanning solution tool will probe the network(s) and then determine if successful communication can be made while keeping time performance logs. Time logging will be used to assess the degree of success produced by the Binderscan implementation.

# 4.1 Network Building Model

Building out networks to test the proposed scanner was essential to developing and collecting time metrics. Networks were planned for class A, B, and C sizes; however, resource [limitations](#_6._Limitations) prevented expansion beyond the standard class C network. The network was built in VirtualBox as a host-only network with randomized IP addresses selected and assigned to hosts running pseudo-randomly selected operating systems supporting differing services. Services include various web applications, databases, simple Linux servers, LAMP stacks, FTP servers, and SMTP servers for diverse attack surface. This randomized build configuration was generated by Python and then deployed using Vagrant. Python was used to create the “VagrantFile.” Vagrant was used for building the test network with randomized values in a systematic and repeatable way from the “VagrantFile.” The randomization ensured the tester did not have foreknowledge of the built network before scanning, which makes it more like a real-world scenario.

# 4.2 Net-Scan Discovery Model

The discovery model is established such that the fastest scanning methods are prioritized. First, ICMP scans are run, response data collected, and the accounting of responsive hosts, if responses are successfully collected, the scan ends. Otherwise, if nothing is found in any given subnet undergoing examination, then asynchronous TCP port scanning runs on the topmost twenty commonly utilized ports, as defined by nmap.org. If still no responses are collected by Binderscan, the scanner will initialize a slower, more in-depth top one hundred Nmap SYN scan. Lastly if no responses have been collected for any given subnet, an asynchronous UDP scan is run on the top twenty UDP ports. Results are stored in a strategically designed dictionary and then written to a CSV file in ascending order by responsive host count. The reporting makes it clear which network subnets could not be reached at the top of the results. This quickly makes access verification brief and indisputable. Simply flipping the “host count” column within excel will put networks with the most responsive hosts at the top of the list. With responsive host counts ordered in this way, the determination of proper segmentation is instantly apparent. Segmentation is not configured correctly if any host count is greater than zero.

# 4.2.1 Background

Pen Testers have various degrees of work experience with many potential focuses, such as Wireless, IoT, Cyber-Physical Systems, Cloud, Web Applications, API, Mobile, and Computer Networking. This variety of possible backgrounds lends itself to any given tester having little to no experience with the computer networking focus. The lack of experience and networking focus makes it difficult to use much less refine and optimize scanning tools for speed and effectiveness. These factors show how automated guard rails are needed to assist these testers with access and segmentation testing.

# 4.2.2 Probing the Network

The tool developed takes advantage of the best open-source scanning tools for computer network probing. The tools selected are Fping, UnicornScan, and Nmap. All these tools are renowned, especially Nmap, given the involvement of Google in its continued development. The proposed scanning tool, developed for this project, orders scanning from fastest to most thorough. The scanning is done in four stepped phases:

1. Phase 1 – Ping Sweeping
2. Phase 2 – Asynchronous TCP Scanning of Top Twenty Most Commonly Used Ports Only
3. Phase 3 – Slower TCP Port Scanning on the top one hundred most used ports
4. Phase 4 – Asynchronous UDP scanning on the top twenty most used ports

Prioritizing the scan methods by the fastest down to the most thorough, followed by a “Hail Mary” UDP scan. Stepping these scan methods in phases this way was intended to reduce required scan durations in most cases. In addition, a network size evaluation occurs in the code to determine whether the proposed scanner needs to use an IP sampling from the IP pool. The details of the sampling methodology can be found in Section 4.1.4 of this report.

# 4.2.3 Scanner Tuning

As mentioned in the prior section, the underlying scanners selected are Fping, UnicornScan, and Nmap. Each tool was tuned for optimal performance.

Phase one Fping scanning was tuned to the following command:

|  |
| --- |
| $>fping -4 --addr -r 1 -a -i 1 {ips} |

**Flags used for performance include:**

|  |  |
| --- | --- |
| **Flag** | **Description** |
| -4 | Restrict target IPs to IPv4 addresses |
| -r | Retry limit (default 3). The number of times an attempt at pinging a target will be made, not including the first try |
| -i | The minimum time (in milliseconds) between sending a ping packet to any target (default is 10, the minimum is 1). |

Phase two, UnicornScan TCP scanning, was tuned to be the following command:

|  |
| --- |
| $>unicornscan -mT -r200000 -L 2 {ip\_sample\_list} --ports {top\_20\_ ports\_list} |

**Flags used for performance include:**

|  |  |
| --- | --- |
| **Flag** | **Description** |
| -mT | TCP Syn Scan Only |
| -r | Rate in PPS |
| -L | Packet timeout delay |
| --ports | Top 20 ports only |

Phase three, Nmap scanning, was tuned to the following command:

|  |
| --- |
| $>nmap –disable-arp-ping -Pn --open -T5 -sS -n --exclude {scanner\_ip} --top-ports 100 |

**Flags used for performance include:**

|  |  |
| --- | --- |
| **Flag** | **Description** |
| --disable-arp-ping | Disable ARP Scans |
| -Pn | Skip ping sweeps |
| -T5 | Run at max speed |
| -n | No reverse DNS look ups |
| --top-ports | Only scan top ports |
| -sS | Syn Scan Only |

Phase four, UnicornScan UDP scanning, was tuned to the following command:

|  |
| --- |
| $>unicornscan -mU -r200000 {ip\_sample\_list} --ports {port\_list} |

|  |  |
| --- | --- |
| **Flag** | **Description** |
| -mU | UDP Scan Only |
| -r | Rate in PPS |
| --ports | Select ports only |

All tooling was configured with speed in mind. Scanners that require PPS rates were set to 200,000 packets per second which is 2/3rds of the maximum theoretical throughput from a virtual machine (Graham).

# 4.2.4 Sampling Method

In addition to fine-tuning the scanners for optimization, subnet size sampling was used. Sampling effectively selects a subset of IP addresses from the scanned IP subnet range. Sampling reduces the time cost by reducing the addresses to check. Each subnet was examined in accordance with its size. IP address samples are taken randomly based on the network class being scanned. Class C networks were not sampled for ICMP scans but instead scanned in their entirety due to the ~2.5-second time cost. However, only 30% of the IP space was scanned in Class C networks for TCP and UDP scans. Class B networks are set to scan only 10% of the IP space.

In contrast, Class A networks are set to scan .005% of their IP space. Sampling larger IP spaces reduce time expenditures in the scanning process. The selection occurs by generating all IP addresses in the CIDR(s) provided and then randomly selecting the associated percentage of IPs from that range.

# 4.2 .5 Acquiring Metrics

Binderscan was altered to include timing from the beginning of a scan to the end of a scan. The metrics stored include the date and time of the scan tool run, the CIDR range(s) scanned, the network class examined, and the scan duration for each run. The scans were run on a class C network and recorded accordingly. These metrics collected will be leveraged to evaluate the tool’s efficiency.

Additional scans were run with the Nmap’s basic usage and the related work developed by Yozer Esneider Garcia Marulanda, named “El Segmentador,” all done on the same network. This data collection allowed for metric comparisons from which many determinations were made and can be found in the evaluation section of the report (See [Section 5](#_5_Evaluation)).

# 5 Evaluation

Binderscan was evaluated by determining the speed of scanning outcomes. The rate of scans largely depended on the size of the network(s) being scanned and the services enabled on that network. In evaluating the Binderscan tool, network scans were done against a Class C network under various conditions. The conditions are different, with each network build containing different numbers of active machines running an array of various services. Also, in some cases, ICMP is disabled, and sometimes it is enabled. The ICMP configuration depends on the network of the examined subnet(s). The time trails reflected the speed of the Binderscan compared with the “El Segmentador” tool discussed in the related works section of this report, as well as Nmap with basic usage. The rest of the evaluation subsections will cover the results of these time trial activities in terms of performance.

In addition, the scanner resides on a virtual machine running Kali Linux OS and is provisioned with 8 GB of RAM, 2 CPUs, and 80 GBs of disk space. The code contains three hundred eighty-seven lines of code, twelve libraries, one class, and six functions on top of the code’s logic.

# 5.1 Performance

There are many variables to the speed outcome of various ping sweep and port scan executions. Class C networks have 256 potential hosts. The Binderscan tool was tested first on a class C network, and the trial runs were repeated one hundred times but only thirty times for the other slower tools, and only 30 times for all tools where ICMP was disabled. The number of runs or executions was selected as thirty data points is generally accepted as a statically valid while one hundred runs is ideal. The first set of one hundred runs was with ICMP enabled on the network. The binderscan.py sc successfully checked the network and determined reachability in roughly one second on average, see Figure 1.

Graphical user interface, text, application, email

Description automatically generated

Figure 1 – Binderscan - 100 Executions - Class C (ICMP Enabled)

Scanning with Binderscan against a Class C network, where the network blocked ICMP, yielded slightly slower results. The slower time was because port scans were required. The Binderscan script strategically utilizes an asynchronous port scanner which is very fast, firing off 200,000 packets per second.

Ultimately with ICMP blocked, Binderscan took an average of 174.85 seconds (~2 minutes) to complete.

Graphical user interface, text, application, email

Description automatically generated

Figure 2 - BinderScan - 100 Executions - Class C (ICMP Disabled) – Line Graph

For comparison, the “El Segmentador” tool took an average of 983 seconds (~16 minutes) per execution against the same Class C network under the same conditions. El Segmentador was passed the requested inputs upon execution, including the directory for output, the network interface to use, and the file containing the IP CIDR range targets. The command to run El Segmentador thirty times required the following command:

|  |  |
| --- | --- |
| for i in $(seq 1 31);do printf "outfiles\neth0\nnetwork\_scope.txt" | ./el\_segmentador.sh;done |  |

The outcome of the El Segmentador scan is made digestible in Figures 3 and 4.

*Chart, line chart

Description automatically generated with medium confidenceFigure 3 - El Segmentador - Class C (ICMP Enabled) – Line Graph*

Graphical user interface, chart, line chart

Description automatically generated

Figure 4 - El Segmentador - Class “C” Network – ICMP Disabled

An additional comparison was made with Nmap running alone with basic usage.

Here is the Nmap command:

|  |
| --- |
| $>nmap --disable-arp-ping -sT -sU 192.168.3.0/24 |

|  |  |
| --- | --- |
| **Flag** | **Description** |
| -disable-arp-ping | Disables ARP Scan |
| -sT | TCP Scan |
| -sU | UDP Scan |

With ICMP enabled, Nmap ran to completion on average at nearly 19 minutes.

Chart, line chart

Description automatically generated

Figure 4 - NMAP Only ICMP Allowed Executions

Then with ICMP disabled, Nmap ran to completion for an average of ~19 minutes.

Chart, line chart

Description automatically generated

Figure 5 - NMAP Only ICMP Disabled – Line Graph

Reflecting upon the results from the Class C network scans where ICMP was enabled, it is clear that Binderscan ran ~614 times faster than Nmap alone and ~613 times faster than El Segmentador, as is apparent in Figures 6 and 7.

Chart

Description automatically generated

Figure 6 - Class C - Tool Comparison (ICMP Enabled) – Bar Graph

Chart

Description automatically generated with medium confidence

Figure 7- Class C Tool Comparison - Line Graph

When ICMP is disabled, Binderscan runs ~6.5 times faster than Nmap and ~6.1 times faster than El Segmentador, as witnessed in Figures 8 and 9.

Chart

Description automatically generated

Figure 8 – Class C - Tool Comparison (ICMP Disabled) – Bar Graph

Chart, line chart

Description automatically generated

Figure 9 - Class C - Tool Comparison (ICMP Disabled) Line Graph

More extensive computer networks will be accounted for via projections. Projecting the findings from Class C scans onto Class B and Class A networks. A single Class B CIDR /16 network contains 256 Class C CIDR /24 networks. Predictions were made using the proper multiplier above for scaling to a class B network in Figure 8 and Figure 9.

Chart

Description automatically generated

Figure 10 - Class B Projections (ICMP Enabled) – Bar Graph

Table

Description automatically generated

Figure 11 - Class B Projections (ICMP Enabled) - Line Graph

Then projections for Class B network scanning with ICMP disabled are shown in Figures 12 and 13.

Chart

Description automatically generated

Figure 12 - Projection of Class B (ICMP Disabled) – Bar Graph

Chart, line chart

Description automatically generated

Figure 13 - Class B Projections (ICMP Enabled) – Line Graph

A single class A CIDR /8 network contains 65,536 class C CIDR /24 networks. Projections were made using this multiplier. The projections where ICMP is enabled can be observed in Figures 14 and 15.

Chart

Description automatically generated

Figure 13 - Class A Projection (ICMP Enabled) – Bar Graph

Chart, table, line chart

Description automatically generated

Figure 14 - Class A Projection (ICMP Enabled) - Line Graph

Projections with ICMP disabled can be observed in Figures 16 and 17.

Chart

Description automatically generated

Figure 156 - Class A Projection (ICMP Disabled) – Bar Graph

Chart

Description automatically generated with medium confidence

Figure 17 - Class A Projection (ICMP Disabled) - Line Graph

The main observation from the projections is that as a network’s size increases, the time savings from Binderscan increases.

In addition to the scan speed improvements, Binderscan removes the need for manual cross-scan analysis, thus reducing the time sink even further as witnessed in Figure 16 below.

Graphical user interface, application, table, Excel

Description automatically generated

Figure - Binderscan Outfile Example

# 6. Limitations

During experimentation, a couple of limitations emerged. Limitations were discovered around the possible sandbox sizes and the scanners leveragedpower. The sandbox size shortcoming popped up when attempting to expand from a class C network to anything more significant. The tool known as Vagrant (IaC) was used to build the class C network sandbox, which functioned as expected. However, attempting to create a class B network with Vagrant with even a small amount of IP space utilization resulted in VirtualBox virtualization software overloading and crashing. To build a class B network for testing purposes, Terraform and AWS were utilized; however, the AWS platform limited EC2 instances to merely thirty-two hosts. In addition, the originally selected asynchronous scanner known as Masscan could not run in the sandbox environment due to the lack of a router. Masscan requires access to a router to run successfully. Since the VirtualBox host-only environment uses only virtual switching, and no router is accessible, the project code was adjusted to use UnicornScan. UnicornScan is yet another asynchronous scanner that is less well-documented. While the scanner swap was not desired, it appears to have been a one-for-one switch, as the rate setting in both tools accepted 200,000 packets per second.

# 7. Future Work

Continued work on network scan efficiency stands open for additional refinement regarding the number of selected ports and tuning of sample sizes to use based on network class. The community needs to work to map scan tools into python libraries for ease of use and to have a monolithic set of dependencies that can all be addressed with pip. Additionally, I would like to see tools like the one proposed wrapped up nicely into a Docker container to save the user from any underlying system-specific or tool-specific dependencies.

# 8. Conclusion

In conclusion, Binderscan, leveraging a four-phase discriminate sequential approach along with IP pool sampling and fine-tuning of underlying scanners, ultimately afforded performance improvements upward of ~99% where ICMP was enabled. Performance improvements of ~85% where ICMP was disabled. These are significant performance improvements concerning scan speed. In addition to the rate of scan times, the tool’s output also allows for expedited analysis. The Binderscan deliverable eliminates the need for manual cross-scan out-file analysis. The time savings from the automated CSV output from Binderscan for access confirmation and segmentation testing is incalculable.

# Acknowledgment

I want to thank the Professor, Staff, and my esteemed team members for their invaluable feedback throughout the course. I want to thank the Slack and Discord communities focusing on Kubernetes, Docker, Vagrant, Terraform, and AWS for pointing the experiment in the right direction to address the sandbox construction of this experiment and more.

# References

Ablon, Jared. “How to Go beyond PCI Compliance to Secure Your Organization: Introduction.” *Medium*, The Startup, 22 Jan. 2020, https://medium.com/swlh/how-to-go-beyond-pci-compliance-to-secure-your-organization-introduction-bf154a33c822.

Baykara, Surkay. “What Are the PCI Compliance Fines and Penalties?” *PCI DSS GUIDE*, 1 Jan. 2022, https://www.pcidssguide.com/what-are-the-pci-compliance-fines-and-penalties/.

Graham, Robert David. “Masscan/Readme.md at Master · Robertdavidgraham/Masscan.” *GitHub*, 26 May 2021, https://github.com/robertdavidgraham/masscan/blob/master/README.md.

Guide, PCI Compliance. “PCI Compliance Guide Frequently Asked Questions: PCI DSS Faqs.” *PCI Compliance Guide*, 5 Sept. 2017, https://www.pcicomplianceguide.org/faq/#1.

Hendricks, Steve, et al. “Inside Target Corp., Days after 2013 Breach.” *Krebs on Security*, 21 Sept. 2015, https://krebsonsecurity.com/2015/09/inside-target-corp-days-after-2013-breach/.

IP Address Class, IP Address Class. “IP Address Class A, B and C Network and Host Capacities.” *The TCP/IP Guide - IP Address Class A, B and C Network and Host Capacities*, Sept. 2005, http://www.tcpipguide.com/free/t\_IPAddressClassABandCNetworkandHostCapacities.htm.

Jones, Corrin. “Warnings (& Lessons) of the 2013 Target Data Breach.” *Red River | Technology Decisions Aren’t Black and White. Think Red.*, 3 May 2022, https://redriver.com/security/target-data-breach.

Louis, Jack. “Unicornscan(1) - Linux Man Page.” *Unicornscan(1) - Linux Man Page*, 2004, https://linux.die.net/man/1/unicornscan.

Lyon, Gordon. *Nmap Network Scanning*. Insecure.Com LLC, 2010.

Maślewski, Piotr. “Dockerize Your Python Command-Line Program.” *Medium*, The Startup, 26 Oct. 2020, https://medium.com/swlh/dockerize-your-python-command-line-program-6a273f5c5544.

Marulanda, Yozer. “Yozgarcia/el\_segmentador: Tool to Automate Corporate Network Segmentation Test.” *GitHub*, 6 Aug. 2020, https://github.com/yozgarcia/el\_segmentador.

McNab, Chris. “Network Security Assessment, 2nd Edition.” *O’Reilly Online Learning*, O’Reilly Media, Inc., 2022, https://www.oreilly.com/library/view/network-security-assessment/9780596510305/ch04.html.

Moldes, Christian. “Compliant but Not Secure: Why PCI-Certified Companies Are Being Breached.” *CSIAC*, CSIAC, https://csiac.org/articles/compliant-but-not-secure-why-pci-certified-companies-are-being-breached/.

PCI, Counsel. “PCI Security Standards Council.” *Payment Card Industry Data Security Standard*, Mar. 2022, https://www.pcisecuritystandards.org/documents/PCI-DSS-v4\_0.pdf.

Saive, Ravi. “Fping - a High Performance Ping Tool for Linux.” *Fping – A High Performance Ping Tool for Linux*, 31 July 2018, https://www.tecmint.com/ping-multiple-linux-hosts-using-fping/.

Schemers, Roland. “FPING.” *Fping Man-Page*, 1992, https://fping.org/fping.1.html.

Weinberg, Neal. “DHCP Defined and How It Works.” *Network World*, Network World, 14 Mar. 2022, https://www.networkworld.com/article/3299438/dhcp-defined-and-how-it-works.html#:~:text=When%20a%20device%20wants%20access,when%20the%20device%20shuts%20down.

Woock, Kurt. “What Is PCI Compliance? 12 Requirements & Guide.” *What Is PCI Compliance? A Guide for Small-Business Owners*, 18 Oct. 2022, https://www.nerdwallet.com/article/small-business/pci-compliance#:~:text=Every%20merchant%2C%20regardless%20of%20the,PCI%20Security%20Standards%20Council.

# Appendix A

To access the code for Binderscan, the build scripts, and all logs from this project, please navigate here: <https://github.com/jsmit260/Binderscan>