University of Zurich^UZH

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Traffic Sinkhole for Cyberattack Analysis

Kyrill Hux
Zürich, Switzerland
Student ID: 17-936-550

Supervisor: Jan von der Assen, Muriel Franco
Date of Submission: March 27, 2022

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis

Communication Systems Group (CSG)

Department of Informatics (IFI)

University of Zurich

Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

URL: http://www.csg.uzh.ch/

Zusammenfassung

Diese Arbeit befasst sich mit dem Entwurf und der Implementierung eines Netzwerkverkehrs-Sinkholes, das in die bestehende SecGrid-Plattform der Universität Zürich integriert ist. Das Hauptziel dabei war eine einfache Erkennung und Umleitung von bösartigem, von Malware stammendem Datenverkehr zu ermöglichen. Nach einem Überblick über bestehende Lösungen und Ansätze zur Verkehrserkennung und -umleitung wird DNS in Verbindung mit einer Blacklist als Ansatz für diese beiden Probleme aufgrund seiner geringen Eingriffsintensität und einfachen Bereitstellung gewählt. Ein vollwertiges, leicht vom Benutzer konfigurierbares DNS-Sinkhole wird dann als Teil dieser Arbeit implementiert. Es bietet eine grafische Benutzeroberfläche mit vielen Optionen für den Benutzer, darunter die Möglichkeit, die Blacklist auf verschiedene Arten und Weisen zu konfigurieren, einschliesslich dem automatischen Herunterladen von einer URL, sowie einen Überblick über die am häufigsten angeforderten Domains auf der Blacklist. Diese Implementierung wird anschliessend auf Leistung und Effektivität bei der Abwehr verschiedener Malware-Familien getestet. Die Ergebnisse sind insgesamt positiv: Das Sinkhole führt nur zu einer zusätzlichen Latenz von etwas mehr als 2ms, während es gleichzeitig Blacklist-Erkennungsfunktionen bereitstellt. Die meisten getesteten Malware-Familien konnten erfolgreich daran gehindert werden, ihre böswillige Absicht zu erfüllen, mit der bemerkenswerten Ausnahme von Ransomware.

Abstract

This thesis deals with the design and implementation of a network traffic sinkhole integrated into the existing SecGrid platform developed by the University of Zurich, with the main goal of allowing easy detection and diversion of malicious traffic originating from malware. After an overview of existing solutions and approaches for traffic detection and diversion, DNS in conjunction with a blacklist is chosen as the approach for both of these issues for it's low intrusiveness and easy deployability. A full-fledged, easily user-configurable DNS sinkhole is then implemented as a part of this work. It offers a graphical user interface with options for the user to configure the blacklist in various ways, including automatic polling from an URL, as well as an overview of the most requested blacklisted domains, among other features. This implementation is subsequently tested for performance and effectiveness in mitigating various malware families. The results are overall positive: the sinkhole only introduces an additional delay of just over 2ms while providing blacklist detection capabilities, and most tested malware families could successfully be prevented from fulfilling their malicious intent, with the notable exception of ransomware.

Acknowledgments

I would like to thank my supervisor Jan von der Assen for his continuous support and provided insights over the entire duration of the project. Jan's feedback was always helpful never failed to point me in the right direction.

Additionally, I want to thank Prof. Dr. Burkhard Stiller, who leads the Communication Systems Group, for enabling me to write this thesis, but also for always offering his help in times of need.

Lastly, I would like to thank Sina for her neverending support.

Contents

Chapter 1

Introduction

Despite all efforts made by both the research community and industry, cyberattacks pose significant threats to the operation of businesses. This issue is magnified at Small and Medium-sized Enterprises (SME) and enterprises that strongly rely on technology [36]. For example, the Computer Emergency Response Team of the Swiss Government (GovCERT) estimates that 123,756 systems operating within the Swiss IP-address (Internet Protocol) space are vulnerable to be exploited for DDoS (Distributed Denial-of-Service) attacks [42].

1.1 Motivation

Although motivating the need for businesses to understand the presence of vulnerable and infected systems within their IT (Information Technology) landscape is simple, deriving actual insight remains a challenge. Related research considers that cyberattacks such as DDoS attacks are best identified at the target [46]. However, identifying attacks using this destination-based approach reveals only the tip of the iceberg. For example, a DDoS attack may be launched from a number of previously infected servers forming a botnet. While the DDoS attack, including underlying attack vectors, can be detected by an Intrusion Detection System operated by the victim, the C&C (Command & Control) traffic cannot be analyzed. Therefore, the proposal shall follow a visual approach to create insights on infected systems using a source-based traffic analysis approach.

Visualization systems present an efficient approach to detect patterns in network traffic [18]. Such a visualization system was targeted in the first implementation of DDoSGrid by [7]. DDosGrid had the objective of helping the network operator detect DoS attacks by providing a visualized data representation of the network. The detection of DoS attacks still requires manual investigation of network data. DDosGrid 1.0 takes as input the captured network data that has many dimensions so that it can visualize them to the network operator for analysis. This tool also allows users to implement their own visualization methods that can easily be integrated into the system to fulfill specific needs of users.

The DDoSGrid system is agnostic to the location where data is collected. Further, since the first iteration considered post-mortem attack analysis, full packet captures were used as primary data source. This has the advantage that any aspect of a packet can be analyzed programmatically. Hence, there is potential to implement a source-based traffic analysis system that would be able to extract any property of network traffic. Given the focus on post-mortem attack analysis, the platform does not provide any features to divert and record traffic. Integrating these components in a way that allows fine-grained traffic diversion toward analysis platform results in a fully integrated system that allows source-based malware traffic analysis.

1.2 Description of Work

To close the gap in accessible integrated diversion and analysis tools, this work mainly examines the design and implementation of a network traffic sinkhole designed to divert and ultimately mitigate malicious network traffic. In order to lower deployment costs and minimize intrusiveness, DNS (Domain Name Service) was chosen as the diversion method over other approaches such as software-defined networks. After implementing a standalone prototype application, this work covers its integration into the existing SecGrid architecture on both the front- and the back-end of the platform. Originally, SecGrid was developed to visualize existing network capture files to analyze attacks post-mortem. In addition to the aforementioned sinkhole, this works also implements a direct capturing functionality into SecGrid, allowing the two new components to work together seamlessly. The new additions are finally tested for performance and effectiveness in their intended use-cases.

1.3 Thesis Outline

This introductory chapter has outlined the motivation to build an integrated diversion and analysis platform using SecGrid as a basis for this work. The rest of the chapters that compose this work are arranged in the following way: Chapter 2 lays a foundation for the rest of the paper by introducing a typology on malware types and SecGrid as a platform, which the next Chapter then uses to give the reader an overview of related literature on the topic, split into *Malicious Traffic Detection* and *Diversion*. Chapter 4 then describes the necessary changes in the existing SecGrid architecture, the implementation of which is outlined in Chapter 5. Finally, the newly added systems are evaluated on their effectiveness in Chapter 6, before Chapter 7 summarizes and concludes this work.

Chapter 2

Background

In modern times, cyberattacks come in diverse variations: From browser-based JavaScript attacks to social engineering, the scope of attacks can be very broad. However, this work only considers cyberattacks originating from malware installed on a device located inside a local network.

2.1 Malware Types and Their Behavior

Even after narrowing the scope to only include malware attacks, there are many types of malware that need to be considered. Malware variations can include types that do not communicate over the network after infecting a host, such as an offline keylogger which stores its information on the infected device's disk. Due to the nature of this paper, only malware that creates traffic on the network will be considered.

In existing literature, various classifications and taxonomies for malware have been proposed over time, which include both malware that communicates over the network, and malware which does not. Thus, the following sections will only examine families of malware described as *Viruses, Worms, Spyware, Trojans, Ransomware* and *Botnets*. These categories of malware do not describe concrete instances of malware and do not have rigid boundaries. Sometimes multiple categories can even apply to a single malware. Still, these classifications are helpful to differentiate the intent, functions and distribution of certain malware types and will thus be used in following sections.

The term of the computer **virus** has been (mis-)used by many members of both technical and non-technical communities to represent any malware one would come across. Instead of using it as an umbrella term, existing literature suggests using the term for self-replicating programs [26], with some literature requiring a virus to have a host-program in order to run and/or replicate [38].

Worms are usually defined as a special variation of a virus, with the main difference being that worms do not need a host-program in order to self-replicate or run [26][38], but other differences can also include the spread over the network to other devices instead of the viruses' spread on its host computer only [26].

Spyware is a broad term used to describe malware which is used to monitor victims or extract information on them in a hidden manner. This information can stay local in the case of a keylogger that stores the captured data in a file on the victim's local storage, but can also be exfiltrated over the network to a controlling or distributing attacker [13]. Such malware can take on advanced forms such as all-encompassing surveillance tools, often called RATs (Remote Administration Tools) which can be controlled in large numbers by a single distributor and are usually spread as an invisible attachment in modified versions of otherwise legitimate software [27]. In these cases, this classification may also intersect with others, such as the classifications of *Trojans* and *Botnets*, as seen below.

Malware that is disguised as a legitimate program, but also fulfills actions unknown and potentially unwanted by the user can be classified as a **trojan**. This may include the deployment of a botnet client or advanced spyware without the user's knowledge, which will persist on the system even after usage of the original program is terminated [26].

Ransomware is a classification of malware that completely or partially blocks access to certain resources on a victim's system. This may be a form of data, hardware or any other part of the victim's device. Access to this resource is then held to ransom until a certain amount of money is paid. If the ransom is not paid, ransomware may destroy the resources it is holding after a set amount of time, or simply never return access [34].

Networks of compromised computers running the same malware fall under the classification of **botnets**. Usually, these compromised computers unknowingly report to a single operator often called the *Botmaster* and await commands. Depending on the complexity of the malware running on these systems, the botmaster may command their victims to bombard a target with traffic (often called a DDoS attack), download further files onto the victim systems or execute arbitrary commands and/or code on them [14]. Botnets are one of the most significant drivers of online crime by enabling the previously mentioned DDoS attacks, serving as proxies and gateways, or even enabling the further distribution of other malware such as spyware or ransomware onto the infected computers [22].

2.2 SecGrid

SecGrid is an advanced network traffic visualization tool developed by the Communication Systems Group of the University of Zurich [17]. It allows users to upload their existing packet capture files and easily analyze them using various visualization techniques on the fly. While it was originally created with the main purpose being the analysis of DDoS attack traffic data, in its current state, it presents an excellent tool for visual analysis of any kind of malicious traffic and provides insight into potential patterns and connections that would otherwise not be possible.

Chapter 3

Related Work

This work deals with the investigation, implementation and evaluation of DNS-based malicious traffic detection and diversion. However, from a methodological perspective, traffic diversion must be considered in a larger framework. Thus, we discuss existing work as part of an overarching traffic analysis framework as presented in Figure 3.1.

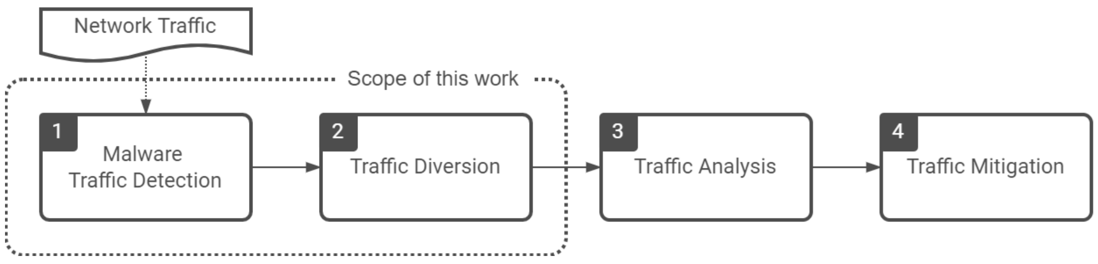


Figure 3.1: Overview of related methodologies

In general, we only consider malicious traffic originating from malware-infected hosts. The following section therefore provides a selection of relevant existing literature that examines traffic detection and diversion approaches of such malicious traffic.

3.1 Malicious Traffic Detection

Most malicious software in modern days produces some kind of network traffic in order to further spread itself, update or modify its current version, or to communicate to a controlling instance. Recent literature has found various ways to detect and identify such traffic and a selection of such approaches will follow in the section below and are summarized in Table 3.1.

A popular, but also one of the most intrusive approaches is to analyze packets and their contents coming through the network and is usually referred to as **Deep Packet Inspection** (DPI). Other than basic signature detection, existing literature has developed advanced ways of detecting unwanted traffic. [24] propose a method of malware packet detection from raw network packets using statistical analysis. They apply Naive Bayes in combination with known signatures to achieve stateless malware traffic detection.

Another approach is shown by [2], where Principal Component Analysis was applied in combination with Artificial Neural Networks in order to classify malware network traffic. The results of this research showed that this approach was able to classify malicious traffic with close to the same accuracy as other state of the art methods of classification [2].

[29] developed a deep learning model which takes raw network data as input and is able to detect and classify malicious traffic contained in it in real time. The paper proposes two versions; one taking raw network packets as its input, and one taking raw network flows as inputs, which would then better be described as *network flow analysis*.

Since DPI deployments need to scan the entirety of all packets coming through the network, powerful professional-grade hardware is needed in order to not slow down the network while maintaining a comparable level of throughput to regular networks, which comes with substantial financial and (training-)time cost.

Network Flow Analysis presents an alternative, less intrusive approach that can be used to analyze network traffic. In contrast to Deep Packet Inspection, no packet contents are analyzed, but rather the describing information around them. This might be information such as the source IP Address, destination IP address, the network interface, and more. One might think of it as the equivalent of a phone bill, not containing the contents of the call, but it can be used to prove who called whom [19].

[41] applied Recurrence Quantification Analysis combined with Machine Learning to network flow data in order to identify malware traffic in the year 2020 and achieved very high accuracy of detection while using fewer flow features than traditional statistical analysis methods.

In their work, [12] used Long Short-Term Memory Recurrent Neural Networks in order to detect botnet traffic originating from internet of things devices. The resulting model scored very high accuracy rates in comparison to other existing models.

The last approach presented in this overview is malicious traffic detection via **DNS**. It presents the least intrusive of all presented approaches and does not analyze any packet contents directly. Instead, a custom DNS server controlled by the analyzing party is installed and all participating hosts have their primary DNS server set to it. The resulting detection techniques operate based exclusively on the DNS traffic coming through the server.

One such detection technique was developed by [5], where a real-time malicious domain detection system was proposed. To achieve this, DNS traffic was collected from a network and then analyzed using 15 unique query features such as various characteristics of the TTL (average, standard deviation and more) or the queried domain names (% of numerical characters, length). According to the authors, this method achieved very positive results and they were able to collect over 100 thousand malicious domain in a public 17 month test-run.

In the industry, the most common practice is to use blacklists comprised of known malicious domains instead of trying to detect malicious domains based on their behaviour as in the previously mentioned work. These blacklists can come in the form of domain block lists such as [1] and [3], or in the form of IP block lists such as [39] or [15] and can be compiled in many ways: some are compiled by hand via user submissions and some come from automatic detection sources such as [5]. Once a machine sends requests to resolve a blacklisted domain or IP, a detection is triggered and further steps can be initiated. This practice is so common in the industry because it has the advantage of removing the burden of detection from each network wanting to use a sinkhole (which would come with its own prerequisites), but also allows for instant detection of already known threats as soon as they appear on the deploying network.

Approach	**Analyzed Data**	**Intrusiveness**	**Prerequisites**
DPI	Entire packets including contents	High (all communication is scanned)	High-power, programmable networking hardware
NetFlow	Network traffic metadata	Medium (only certain network flow metrics are scanned)	Specialized networking equipment
DNS	DNS queries	Low (only domain names are scanned)	Router with configurable DNS server

Table 3.1: Overview of Malicious Traffic Detection Approaches

3.2 Malicious Traffic Diversion

Once malicious traffic has been identified, it is usually desirable that it is diverted from its originally intended path towards a new destination (or no destination at all) controlled by the defending party, in order to prevent that traffic from fulfilling its malicious purpose. Such diversions can be done in various ways, but most existing systems in the industry and recent literature seem to focus on three approaches specifically, all of which are laid out in the following section and summarized in Table 3.2.

The first well-known approach to traffic diversion is through usage of the **Border Gateway Protocol (BGP)**, which refers to a large-scale routing protocol that determines packet routes between and within autonomous systems (AS). As such, it can be used to block or redirect a number of attacks by diverting traffic to another destination than originally intended, often mitigating damage that would be caused by attacks such as DDoS attacks [43]. Using BGP to defend against such attacks is therefore well-documented [8], but research has moved on to various extensions and variations of BGP due to security [10] and granularity [21] concerns. It must be noted that there are a number of prerequisites that must be met in order to use BGP, most notably an organization wishing to employ BGP needs to be in possession of an *AS Number* which can only be obtained from one of the Regional Internet Registries. Currently in Europe, obtaining this number comes with its own set of requirements such as the network being multi-homed (connected to multiple ISPs) [11], effectively restricting access to larger organizations.

Software Defined Networks (SDN) pose the second approach for traffic diversion discussed in this section. Given special networking equipment and configuration, this technology allows to reroute packets traversing a network in real time based on any number of properties they might have, such as their destination IP, port, the protocol they carry and more, and redirecting them to a new destination instead of their originally intended target, or not continue routing them at all. While SDN can be effective for traffic diversion, it is important to consider the cost that comes with it: Specialized (usually enterprise-grade) SDN-compatible networking equipment must be acquired and configuration of such networks requires special expertise in that area.

[4] implemented a simple dynamic traffic diversion algorithm and tested for performance using SDN in a real network (instead of usually emulated networks). They found that their SDN setup was able to reduce packet loss in a high-stress situation from 50% to none by diverting the malicious traffic.

An attempt to make SDN-based attack mitigation systems more accessible has been made by [37], who propose an autonomic SDN-based attack mitigation framework. Their proposed framework allows ship personnel without deep security expertise to create rule-sets and policies for traffic diversion using an easy to understand high-level grammar.

The last diversion approach to be mentioned in this section is using **DNS**. In this approach, a custom DNS server is set up to be used by all hosts in a network and subsequently resolves all queries as usual. If a queried domain or IP is detected by any detection mechanism such as the ones described in the above section, it can be resolved to a different, special address instead. This might be the localhost address to completely prevent traffic from leaving the source entirely, or a designated host in the network which can then record the incoming malicious traffic for later analysis instead. Such a host would then be called a DNS Sinkhole [9].

Approach	**Granularity**	**Financial Cost**	**Prerequisites**
BGP	Low (routes only)	High (ASN, multi-homing, and more)	AS Number, multi-homed network, BGP-capable routers
SDN	High (individual packets)	Medium (special networking hardware)	Specialized networking equipment
DNS	Low (domain-level)	Low (single, relatively low-power machine)	Network with configurable DNS

Table 3.2: Overview of Malicious Traffic Diversion Approaches

[25] show a DNS Sinkhole based approach where they create a DNS Sinkhole server with a configuration GUI that allows users to comfortably edit a blacklist that is used for traffic diversion. Their system also collected the raw diverted traffic into files and stored them

based on source addresses for later analysis. Overall, the authors report positive results for their diversion and collection of malicious traffic, but leave the analysis and mitigation to a future analysis.

Chapter 4

Architecture

As laid out in Chapter 3, existing literature largely focuses on either detection, diversion or analysis of malicious traffic, this work will take the opportunity to fill a gap and combine detection, diversion and analysis in one convenient solution: as an addition to the already existing SecGrid visualization tool (see Section 2.2). Due to the intrusiveness and high prerequisites for the other malicious traffic detection approaches as summarized in Table 3.1, this work will leverage DNS in conjunction with a blacklist for its detection approach of malicious traffic. This approach ensures a high detection rate of known up-to-date threats when using a current and frequently updated public blacklist. The logical choice for traffic diversion follows to be DNS as well, since the two approaches can easily be combined and no new networking equipment needs to be acquired for either of them (see Table 3.2). The end goal should then be a functioning DNS Sinkhole integrated into the SecGrid interface, easily configurable by the user with automatic capturing abilities for the redirected traffic for convenient analysis within the same SecGrid interface.

To achieve this goal, some changes and additions need to be made to the existing SecGrid architecture, shown in green in Figure 4.1. Namely, two new modules need to be added to the Data Layer, *(i)* a *DNS Sinkhole Module* and *(ii)* a *Direct Capture Module*. Additionally, in the User Layer, the *(iii) Web-based Interface* will need to be adjusted to reflect the new additions and serve the user with an easy-to-use overview of the sinkhole configuration.

The **DNS Sinkhole Module** will serve as a DNS server, taking care of incoming requests and resolving them using a real, pre-configured DNS server such as Google's DNS servers. However, it will also keep a blacklist of domains and IP-addresses. There are two ways of triggering a blacklist entry: The first way is to simply query a blacklisted domain. The second is to query a domain that will resolve (e.g. by being a CNAME entry) to a blacklisted domain, or if the blacklisted domain resolves to a blacklisted IP address. If such a trigger occurs, the DNS module will resolve it to a pre-defined sinkhole address (usually the SecGrid host machine's address) instead of the original result. This will result in potentially malicious traffic being diverted from its original destination and towards the newly created sinkhole.

This incoming traffic will then be captured by the **Direct Capture Module**. This module will allow the user to enable capturing traffic directly from a network interface attached

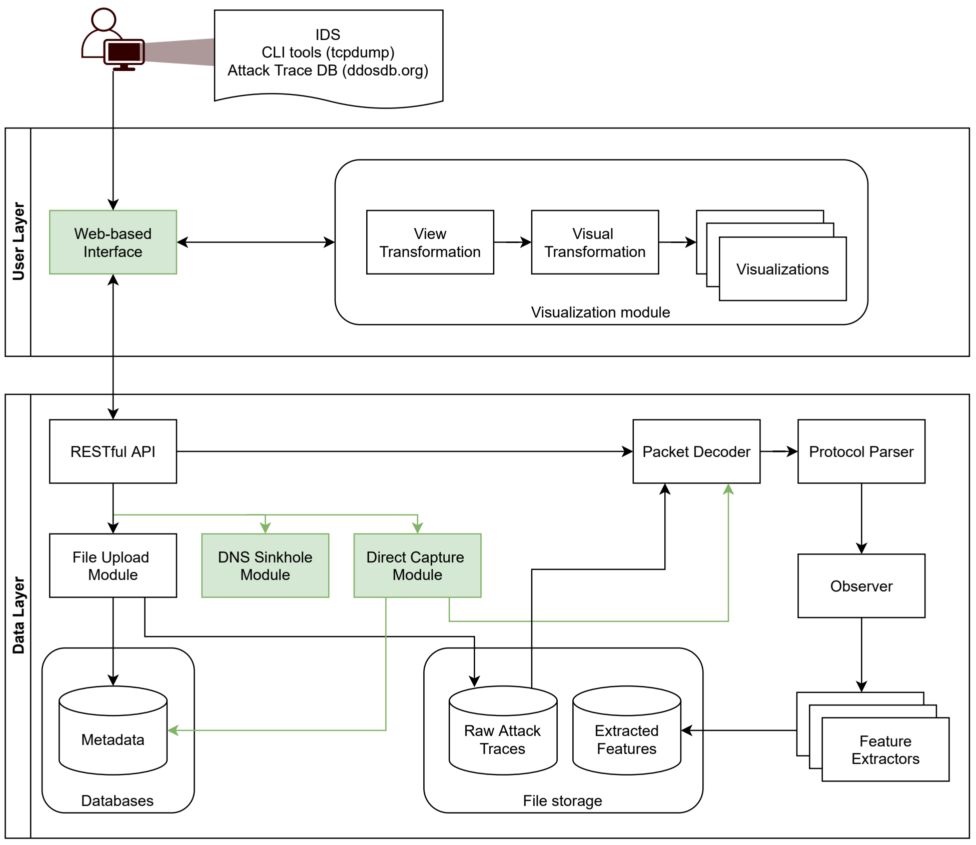


Figure 4.1: Overview of the existing SecGrid architecture and the planned changes to it in green.

to the machine running SecGrid and analyzing it right away. Once the capture is stopped, it can then comfortably be analyzed in the SecGrid UI (User Interface).

Finally, the **Web-based Interface** SecGrid already possesses will be expanded to incorporate two new additions: Firstly, a new interface enabling a user-friendly configuration of the DNS Sinkhole. There, the user will be able to start and stop the sinkhole, as well as to edit the blacklist(s) currently in use. Secondly, a new direct capture interface will be added, allowing users to start capturing from a network interface of their choice, as well as seeing and stopping currently running captures.

Chapter 5

Prototype and Implementation

After the previous chapters defined the general architecture of the new modules, this chapter outlines the implementation process on a technical level. First, the implementation of a proof of concept of the DNS module is described, then the following sections proceed with its integration into the existing systems of SecGrid.

5.1 DNS Module Proof of Concept

To demonstrate the feasibility of the DNS module presented in the previous chapters, a proof of concept (PoC) was developed first. Since the existing SecGrid architecture already used Node.js [16] for the back-end (see "Data Layer" in Figure 4.1), the PoC was also implemented in a standalone application using the same technology.

```
# this line will be ignored!
evil.com
malicious.com
192.168.13.37
```

Figure 5.1: A simple three-entry example blacklist.

The first step in realizing the concept was to create a working, interceptable DNS server using JavaScript. To achieve this goal, the node package `dns2` [40] was used that is well-suited for the purpose of this work, because it provides a low-level, dependency-free hackable DNS client and server implementation. Using the provided building blocks, a first, basic DNS server was implemented that would relay any request to an existing external DNS server (such as Google's `8.8.8.8`) using the DNS client part of the package, receive its response and send it back to the requesting client. Once that was implemented, work on the blacklisting functionality could be started. For easier maintainability and to allow for an easier future integration in the SecGrid architecture, the existing code was first encapsulated in an ES6 (EcmaScript 6) class, which would take relevant parameters such as the relayed external DNS server (henceforth called *mainDNS*) or the port the DNS server should be listening on, but also other important information such as the blacklist itself.

In the PoC, the blacklist was implemented using a simple text-based list, where each line represents one entry of the blacklist in the form of a domain name, or an IPv4 address, as shown in Figure 5.1. The PoC would parse this list into a JSON-array (JavaScript Object Notation) on application startup and hold it in memory for further use. It was decided to hold the entire blacklist in memory for two reasons. Firstly, performance: Searching operations are the fastest when done in-memory, and DNS lookup performance may be an important factor for potential users. Secondly, size: A DNS blacklist should not surpass a size that could realistically be loaded into memory, since extremely large blacklists will always lead to a slowdown of lookup operations and should thus be kept to a manageable size. A similar approach can be seen in existing DNS Sinkhole solutions. For example, one of the most popular existing open source DNS sinkholes, *Pi-hole* [35], by default uses a crowd-sourced block-list that is just under 3 MB in size [6].

```
$ dig @192.168.13.46 -p5333 google.com

; <<>> DiG 9.16.23 <<>> @192.168.13.46 -p5333 google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44871
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                    IN        A

;; ANSWER SECTION:
google.com.            10        IN        A        172.217.168.14

;; Query time: 13 msec
;; SERVER: 192.168.13.46#5333(192.168.13.46)
;; WHEN: Sat Jan 29 21:42:53 CET 2022
;; MSG SIZE  rcvd: 65
```

Figure 5.2: Requesting a non-blacklisted domain (google.com) yields a real resolved response.

Blacklist functionality in the PoC works as outlined in Figure 5.4 and is described in the following section. When a client request is received, a quick lookup in the blacklist is performed first. If the requested domain is contained in the blacklist, the request will be answered as a DNS A record entry pointing to a previously set *sinkhole address*, such as the address of the machine currently running the PoC, or an address like localhost. If the requested domain does not fall in the blacklist, the entire request is relayed as-received

```
$ dig @192.168.13.46 -p5333 evil.com

; <<>> DiG 9.16.23 <<>> @192.168.13.46 -p5333 evil.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1483
;; flags: qr rd ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;evil.com.                        IN        A

;; ANSWER SECTION:
evil.com.               300        IN        A        192.168.13.46

;; Query time: 3 msec
;; SERVER: 192.168.13.46#5333(192.168.13.46)
;; WHEN: Sat Jan 29 21:43:06 CET 2022
;; MSG SIZE  rcvd: 50
```

Figure 5.3: Requesting a blacklisted domain (evil.com) returns the pre-configured sinkhole address `192.168.13.46`.

to *mainDNS*. When a reply is received, it is first checked for further domain matches, such as in the case that the requested domain resolves as a CNAME-entry and points to a different domain, which might in turn be blacklisted itself. This way, quick domain-hopping can be prevented. Lastly, if the domain resolves to a blacklisted IP address, this address will also be replaced with the sinkhole address. If none of the above applies, the response is sent back as-received to the requesting client.

After implementation, this functionality was tested using the Linux command-line tool `dig` to confirm everything worked as intended. These tests were concluded with positive results as can be seen in Figures 5.2 and 5.3, showing the system working as intended. It is also interesting to note that blacklist resolution was handled relatively quickly with a low response time of only 3 milliseconds. Furthermore, real lookups that were sent to *mainDNS* behind the scenes resolved in approximately 13 milliseconds, which equated to only 4 added milliseconds of response time compared to requesting the same information from *mainDNS* directly (9 milliseconds).

5.2 Implementation of the DNS Module

After successful completion of the PoC, the next step to be taken was its integration into the existing back-end architecture of SecGrid, in order to pave the way for communication with a future front-end interface.

Input: A DNS request coming in from a client on the network.
Output: A DNS response returned to the requesting client.

```
domainBlacklist: list of domains
ipBlacklist: list of IPv4 addresses

handleDnsRequest(request) {
    if (request.domain ∈ domainBlacklist) {
        return response pointing to sinkhole address;
    }
    answers = request resolution for request from mainDNS;
    for (answer a ∈ answers) {
        if ((a.hasDomain and a.domain ∈ domainBlacklist) or
            ((a.hasAddress and a.address ∈ ipBlacklist))) {
            return response pointing to sinkhole address;
        }
    }
    return answers;
}
```

Figure 5.4: Filtering algorithm implemented by the DNS module to handle DNS requests.

5.2.1 Back-End (Data Layer)

Since the PoC was implemented in the form of a ES6 class with only two npm dependencies, migrating it into a new codebase was a quick and simple task. Most of the PoC's implementation could be taken over directly, but some optimizations and additions were made for production use. One such optimization was the switch from arrays holding the blacklist data to JavaScript Sets, since they provide surpassingly better performance for entry lookups. An addition had also been made in the form of basic statistical counting of the current run: Every time a blacklist entry was triggered, this feature registers which entry it was and how often it had been triggered by which source address. This statistic can be retrieved using an additional REST (Representational State Transfer) endpoint and is reset every time the sinkhole is restarted. The configuration and blacklist used by the Sinkhole class was configured to be stored on disk and loaded into memory on application startup. For this storage, since frequent updates are not required and both the blacklist and current configuration are kept in-memory at runtime, simple JSON files were decided on for persistent storage.

In order to control the newly added sinkhole module, new REST endpoints had to be introduced to the data layer of SecGrid. Because the DNS sinkhole functionality should not be accessible by the public, all endpoints described in this section require a valid, authenticated user session in order to be accessed, using the same protection as existing endpoints in SecGrid. New endpoints were implemented for getting and setting the sinkhole configuration which includes settings such as the DNS port, sinkhole address and *mainDNS*, as well as endpoints responsible for starting and stopping the sinkhole. Additionally, one status endpoint which displays a summary of the current state of the Sinkhole was added.

Lastly, endpoints responsible for blacklist management were introduced, but presented more of a challenge for implementation. While managing the blacklist directly by sending a list of entries to be considered should be supported, another important feature to implement was the possibility to send a URL pointing to a blacklist file instead. This feature should then have the option of the back-end polling the same URL automatically in regular time intervals, checking for updates to the list. This is especially important because many public blacklists are crowd-sourced online or are being updated regularly to include the most current threats. In order to fulfill this requirement, the endpoint responsible for updating the blacklist was expanded in functionality. Instead of only accepting a blacklist in the form of a JSON array, more complexity was added to the accepted body schema.

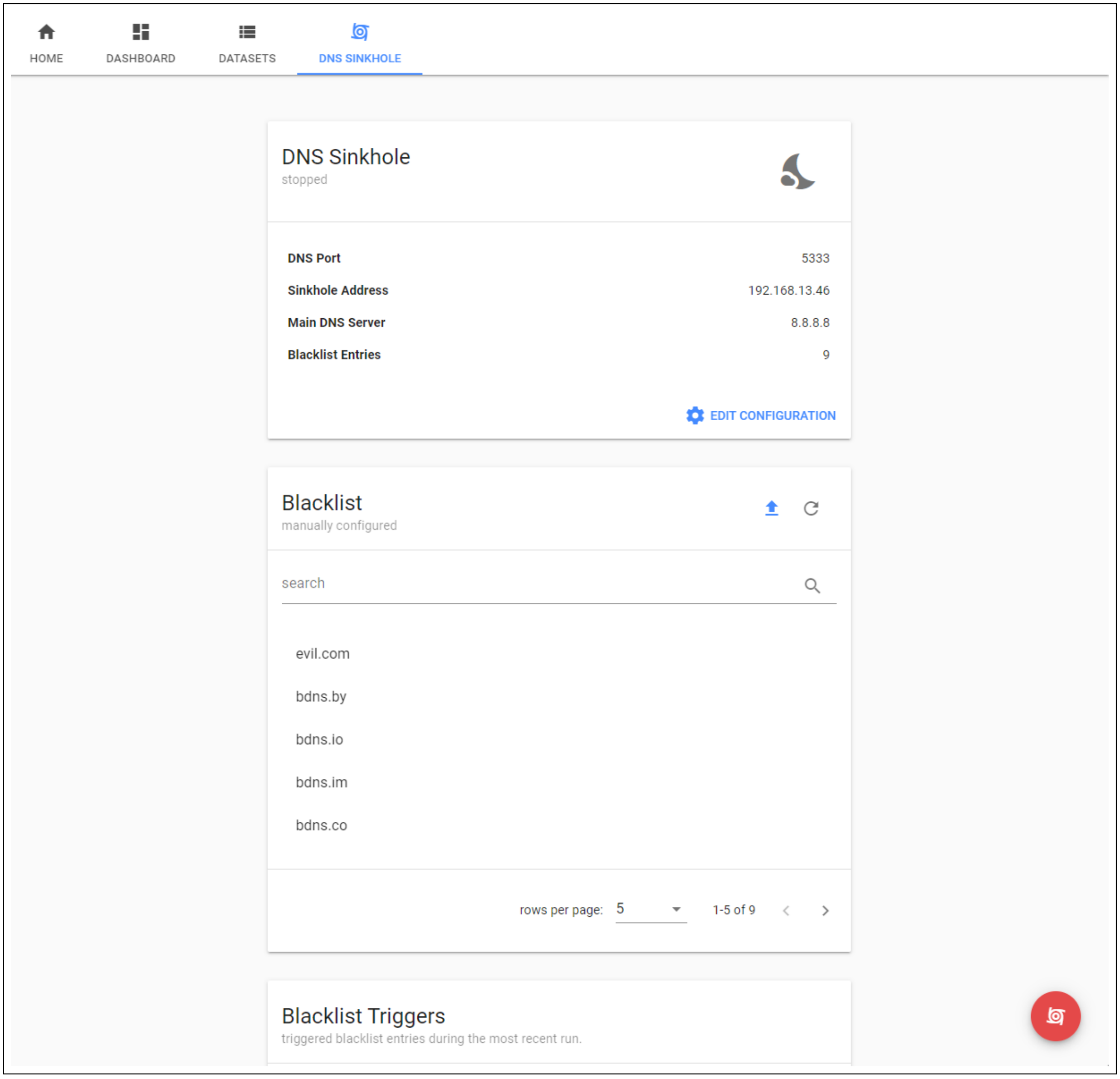
```
// schema A
{
  "url": "https://example.list/blacklist.txt",
  "continuous": false
}

// schema B
{
  "data": ["evil.com", "malicious.com", "192.168.13.37"]
}
```

Figure 5.5: The two schemata accepted by the endpoint for updating the sinkhole blacklist.

As illustrated by Figure 5.5, two schemata have been introduced that are accepted by the endpoint. If the blacklist is to be updated directly, a `data` property containing the blacklist contents in form of a string array can be passed and will be used as the new blacklist. Alternatively, an `url` property containing a direct link to a blacklist file can be passed, which will cause the server to pull the contents and use them as the new blacklist instead. With the latter an additional `continuous` flag can be passed, which will, if set to `true`, enable the backend to issue a `HTTP GET` request to this URL every 60 minutes and update the sinkhole blacklist according to the latest contents of the received file. When a URL is retrieved using either of these methods, any lines starting with a `#` are ignored since many publicly available lists use this format for comments inside of them.

Since client-side URL validation is impossible to do in a secure fashion due to CORS (Cross-Origin Resource Sharing) restrictions (relying on CORS-proxies can open up the application to open proxy or server-side request forgery vulnerabilities), a URL-validation endpoint was also introduced. This endpoint allows clients to send a URL to be tested, and returns whether this URL could successfully be retrieved as well as how many entries could be parsed from it. It should be noted that this and the previously mentioned URL both could pose a security risk by being abused to facilitate SSRF (Server-Side Request Forgery) attacks, causing the server to send `GET` requests to arbitrary URLs, potentially reaching vulnerable internal endpoints inaccessible to the public. But since, as previously mentioned, all endpoints require valid authentication, the risk of external attacks can be concluded to be relatively low.

5.2.2 Front-End (User Layer)



DNS Sinkhole
stopped

DNS Port	5333
Sinkhole Address	192.168.13.46
Main DNS Server	8.8.8.8
Blacklist Entries	9

Blacklist
manually configured

evil.com
bdns.by
bdns.io
bdns.im
bdns.co

rows per page: 5 1-5 of 9

Blacklist Triggers
triggered blacklist entries during the most recent run.

Figure 5.6: The newly implemented Sinkhole dashboard in the SecGrid frontend, with the Sinkhole being in a stopped state.

The new interface for controlling the sinkhole integrated into SecGrid can be reached via a fourth tab in the general navigation bar of SecGrid (see Figure 5.7), which is only accessible once the user has logged in.

When a user clicks this tab, they are then presented with three material design style cards and one floating action button as seen in Figure 5.6 (red). In order to maintain a consistent user experience and design throughout the entire application, it was decided to stick with the material design style and components that were previously used all around the SecGrid front-end.

In addition to the cards, this tab also features a floating action button. This button serves as the main control for the sinkhole, starting it when it's stopped and vice-versa. To clarify its purpose to users, the button shows a vortex icon and the tooltip *start sinkhole* when the sinkhole is stopped, and a stop icon with the tooltip *stop sinkhole* when the sinkhole is running.

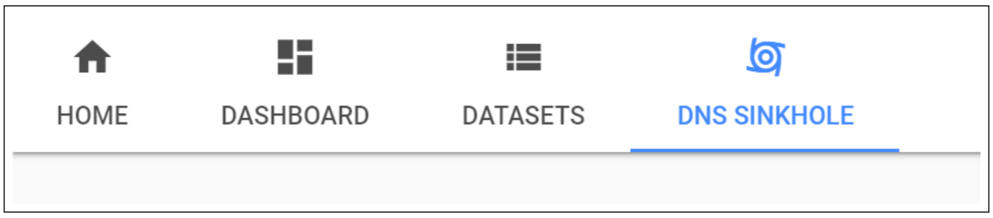
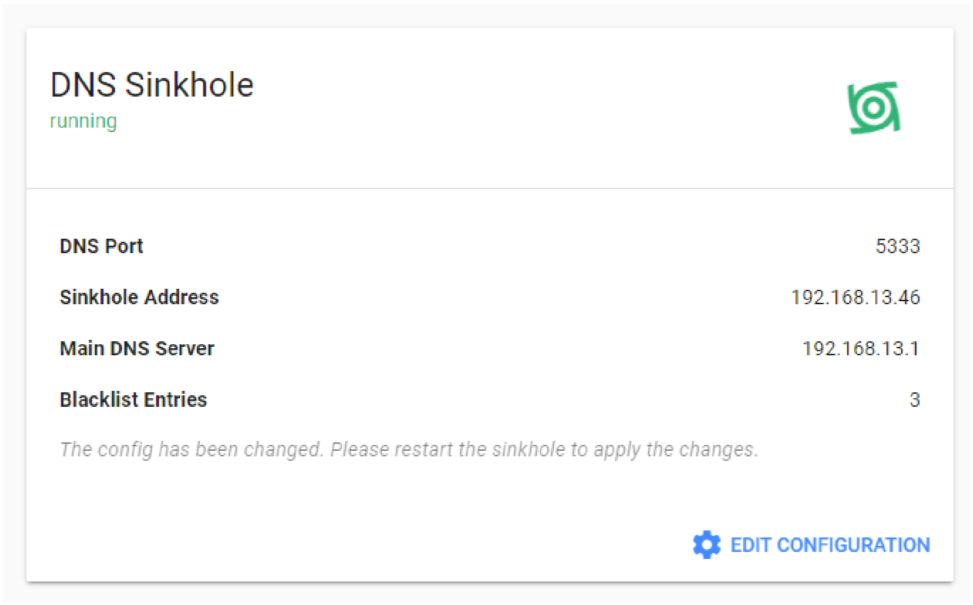


Figure 5.7: The SecGrid navigation bar with the newly added DNS Sinkhole tab.

The Sinkhole Status Card

The topmost card visible in Figure 5.6 shows the general state of the DNS module using the general status endpoint implemented in the back-end and is automatically refreshed every 5 seconds. To make the current sinkhole state visible at a glance, a subtitle was added to the header section of the status card which indicates the state using its text. It also changes colors accordingly, turning green when the sinkhole is running, and turning back to grey when the sinkhole is stopped. Because intial user feedback showed the status to be small and sometimes hard to spot immediately, a large status icon was added on the right side of the card's header section. This icon visually represents the running state of the sinkhole, showing a green vortex when running (see Figure 5.8) and a grey moon when stopped.


DNS Sinkhole
running

DNS Port	5333
Sinkhole Address	192.168.13.46
Main DNS Server	192.168.13.1
Blacklist Entries	3

The config has been changed. Please restart the sinkhole to apply the changes.

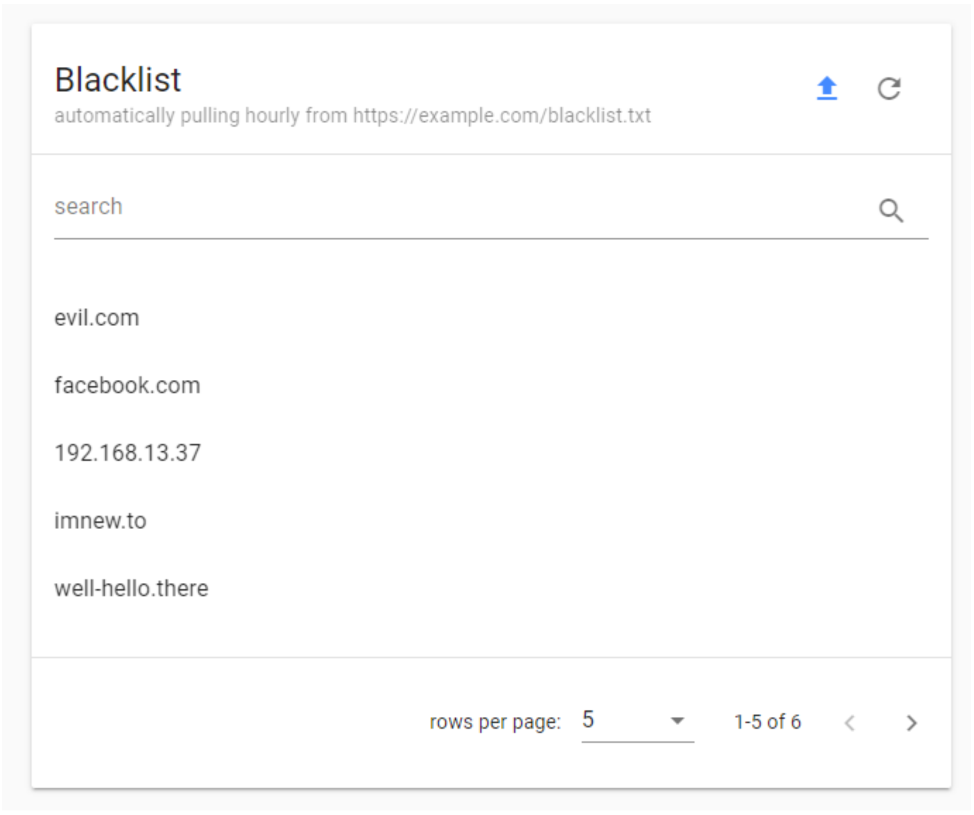
Figure 5.8: The sinkhole status card when the DNS sinkhole is running, but the configuration has been changed.

The card's main body shows an overview of the currently running configuration, including all relevant properties such as the DNS port the sinkhole is listening on, the sinkhole address and the *mainDNS*. It also provides an additional piece of useful information in the form of the amount of currently loaded blacklist entries. If the configuration is changed while the sinkhole is running, a small text will appear under the aforementioned entries, notifying the user to re-start the sinkhole in order to apply the changes (as seen in Figure 5.8).

On the bottom of this card, a single button with the caption *Edit Configuration* can be found. When clicked, it will open a dialog allowing the user to adjust the sinkhole configuration and subsequently saving it. If the sinkhole is running when the configuration is changed, the configuration is only applied on the next start of the sinkhole.

The Blacklist Card

The second card shown in Figure 5.6 displays the currently used blacklist. The header section is comprised of three important elements: Firstly, the title with a dynamic subtitle. This subtitle indicates to the user the currently used blacklist mode. If the user manually uploaded a blacklist, this subtitle will show the text *manually configured*. If a URL has been used to retrieve the currently used blacklist - whether it be hourly or just once, it will be shown accordingly including the URL used (as can be seen in Figure 5.9).


Blacklist
automatically pulling hourly from https://example.com/blacklist.txt

evil.com
facebook.com
192.168.13.37
imnew.to
well-hello.there

Figure 5.9: The blacklist card showing the first five entries of an automatically pulled blacklist.

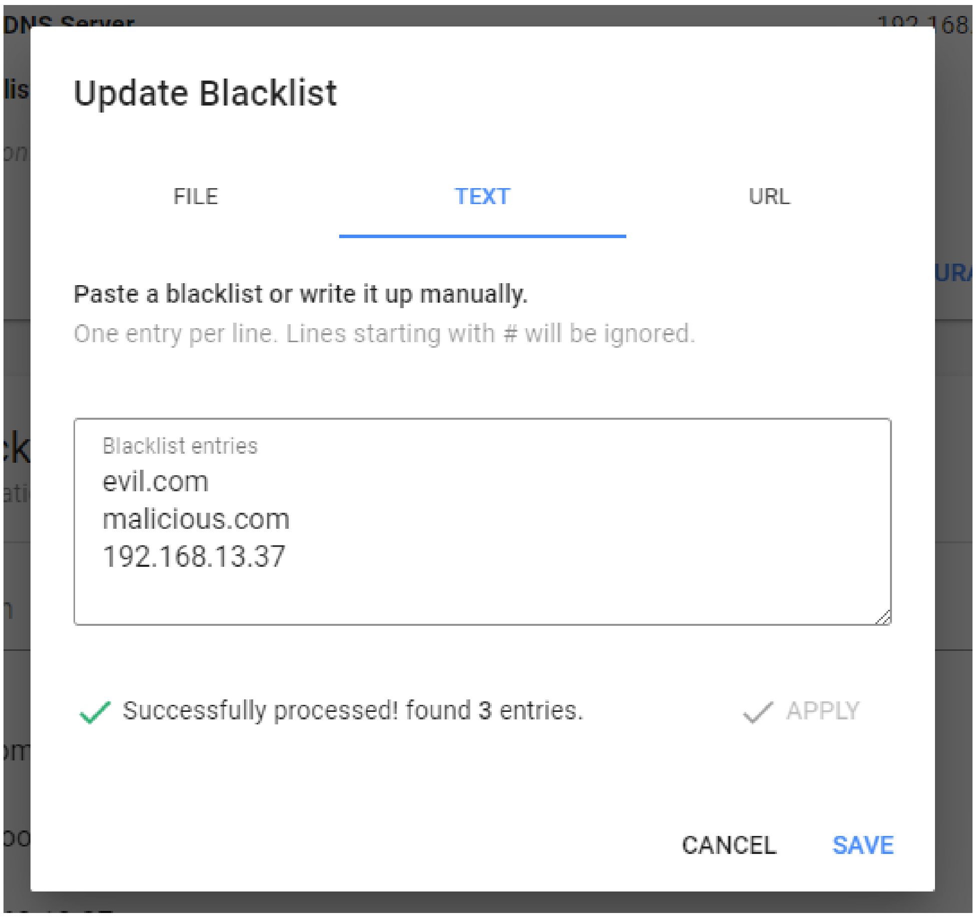
The other two elements housed inside of the card's header are two buttons, both of which only have the form of an icon, but show a tooltip explaining what they do upon hovering, enabling discoverability. The first is a simple refresh button which retrieves a fresh version of the current list for the main body section of the card. The decision to implement a refresh button instead of auto-refreshing the blacklist card's contents in the same fashion as the sinkhole card has been made for two main reasons. Firstly, blacklist contents should not change very often without user input, making frequent refreshes unnecessary. Secondly, blacklists can potentially reach larger file sizes, which could then lead to increased network traffic and resource usage if an entire, very large list is being sent and received every few seconds.

The second button allows the user to update the currently existing blacklist. When clicked, it will show a dialog (shown in Figure 5.10) presenting the user with three different ways of uploading a new blacklist, each contained in its own tab. The first option is to upload a file containing the blacklist entries the user wants to use. The user can select any file and upon selection, the dialog will try to read the file, split it into lines and filter them into respective entries, also leaving out comment lines starting with `#`. If this process is successful, the user is shown a tick mark and a confirmation of how many entries have been found inside the uploaded file, also enabling the *save* button which will, when clicked, upload the result to the back-end and use it as the new blacklist.

The second tab presents the user with a multi-line text box, allowing them to enter their desired entries manually, or paste a prepared text from the clipboard. Once done, an *apply* button can be clicked, which will initiate the same validation process as on file upload and will confirm the user how many entries have been found in the entered data, thereafter enabling the *save* button once again. If any text is changed after processing, the state is reset and the processing must be re-done before saving the blacklist is possible again.

The last tab allows users to upload a blacklist using a URL. After entering a URL, the user must click on an *apply and test* button, which will validate this URL using the validation endpoint previously mentioned in Section 5.2.1, since retrieving data from cross-site URLs is usually infeasible to do from browser-side JavaScript due to security restrictions (CORS policies). Before saving the new blacklist, the user is also presented with a switch, allowing them to decide whether the server should regularly visit the given URL and refresh the blacklist using it's contents. When the save button is clicked on any of the three available tabs, the dialog is closed and the blacklist is sent to the server. When a response is received, the blacklist card automatically refreshes its contents.

These contents are then shown in the main body part of the blacklist card in the form of a searchable, paginated list of entries. Because the main visual components package used in SecGrid, `vue-material`, as of the time of writing does not provide any paginated list or table components, a custom solution was required. Therefore, the search and pagination functionality visible in Figure 5.9 has been implemented from scratch, only using existing components for displaying the current entries and buttons or text-boxes. The pagination feature only shows a limited (but configurable) amount of entries at a time and divides the list into pages with this amount of entries each. Users can then look through these pages individually. For performance reasons, the pagination is done completely on the client-side, meaning that on a refresh the entire blacklist is received and stored in browser



Paste a blacklist or write it up manually.
One entry per line. Lines starting with # will be ignored.

Blacklist entries
evil.com
malicious.com
192.168.13.37

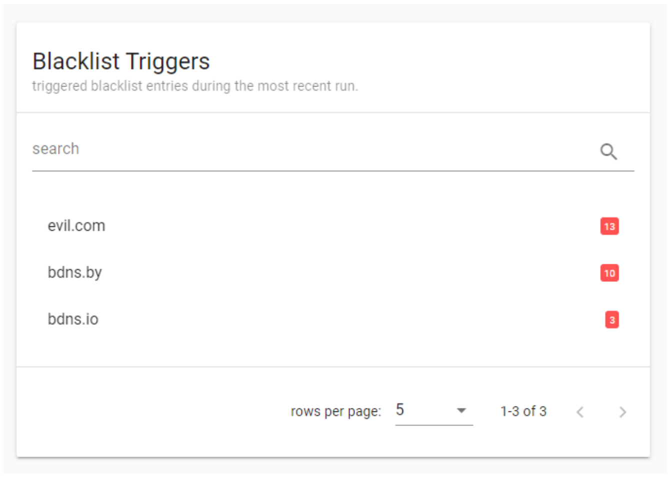
✓ Successfully processed! found 3 entries.

Figure 5.10: The update blacklist dialog's *text* tab which allows the user to manually enter the desired blacklist.

memory. This not only allows users to instantly flip through pages, but also greatly increases search speeds in the search-bar available above the list contents. However, the search does not update in real-time as users type to prevent possible performance bottlenecks with large lists being searched in browsers running on less powerful hardware. Instead, users need to either hit the enter key or click the magnifying glass icon on the right side of the search bar.

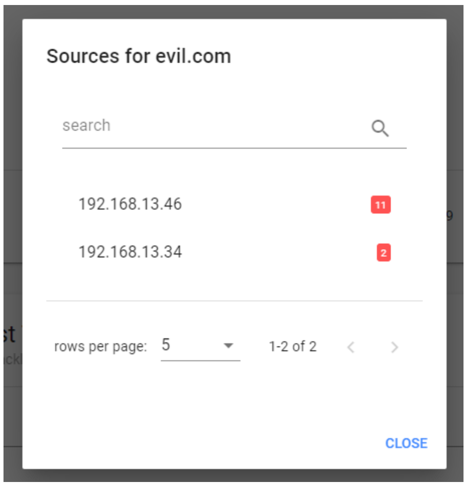
Statistics Card

The third and last card which is barely visible in Figure 5.6 but shown in it's entirety in Figure 5.11 visualizes the simple statistics gathered in the back-end. It presents a continuously updated list of blacklisted domains that have been requested, sorted descendingly by occurrences (shown on the right end of each item). Reusing functionality implemented in the blacklist card, this list is paginated and searchable in the same fashion. Additionally, each of the list's items is clickable and upon click will show a dialog detailing which source IP addresses have tried accessing the blacklisted domains (shown on the right of Figure 5.11). This card provides insight into the usage and effectiveness of the deployed blacklist, and can help administrators find potential threats in their network in a quick and convenient fashion. For example, if the same source address keeps requesting one blacklisted domain in regular intervals continuously over a longer timespan, it will be shown in the statistics card and administrators could then initiate further investigations into a potential infection of this source address.


Blacklist Triggers
triggered blacklist entries during the most recent run.
search

evil.com	13
bdns.by	10
bdns.io	3

rows per page: 5 1-3 of 3


Sources for evil.com
search

192.168.13.46	11
192.168.13.34	2

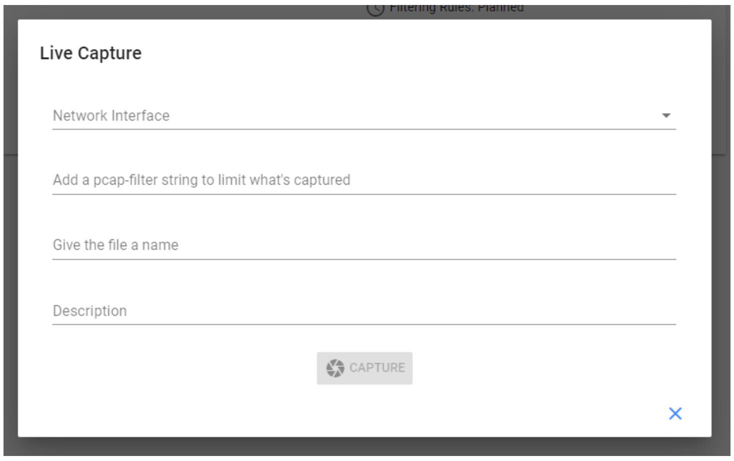
rows per page: 5 1-2 of 2
CLOSE

Figure 5.11: The statistics card (left) showing the most requested blacklisted domains in a sorted list, and the corresponding dialog (right) presenting these request's sources for one entry.

5.3 Implementation of the Live Capture Module

The live capture module allows SecGrid users to capture traffic directly from a network interface instead of capturing it manually using an external tool and importing the capture file afterwards. A simple implementation of a live capture module already existed in one branch of the SecGrid project, therefore this work focuses on improving usability and functionality. The existing solution allows users to start a live capture on a manually typed-in network interface and close it at a later point in time. During the capture, the incoming data is continuously analyzed in order to minimize memory and disk usage and when the capture is closed, the analysis is added as a new dataset to SecGrid. The improvements proposed in this work are twofold: *(i)* The interfaces should be automatically detected and selectable, and *(ii)* since the underlying technology accepts *pcap-filter* [20] strings, the user should have the possibility of entering a filter string when starting a live capture.

The first improvement could be implemented in a relatively simple manner with a single REST endpoint being added to the backend. This endpoint returns a list of network interfaces and their corresponding IP addresses gathered using Node.js' new convenient `networkInterfaces()` method in the included `os` package, allowing the live capture creation dialog to retrieve this list upon being opened and loading it into a `select` component, as seen in Figure 5.12. The selected interface is then simply sent as a string to the backend when the live capture is started, just as in the previous versions and therefore no further changes were needed in that part of the communication.



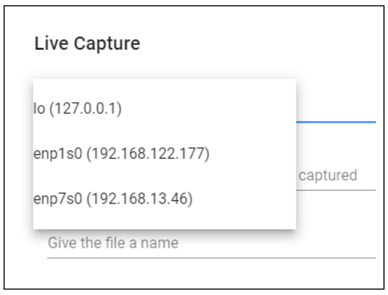


Figure 5.12: The new and improved live capture creation dialog with selectable interfaces and a filter input field.

The second improvement adds a new input in the live capture creation dialog allowing the user to specify a *pcap-filter* string, which can be used to limit the captured traffic in a multitude of ways, for example only capturing incoming ICMP traffic, or only capturing traffic from a certain originating address. This can be very useful for example if there is a known infected machine and the sinkhole user wishes to only capture traffic from that specific host. The underlying implementation in the backend uses a modified version of the *node_pcap* library [33] and calls the `pcap.createSession(interface, options)` method, where the `options` argument can include a *pcap-filter* string that will be used for that capture. But in order to successfully pass the filter string input by the user to the miner that finally creates the live capturing session, multiple steps had to be taken. First, a new field had to be implemented in the endpoint that starts the live capture, simply passing the string that was input by the user. Then, a new CLI option had to be created for the miner sub-application which does the capturing, passing the filter through to it. New parsing code was added on the miner side to read the filter string and finally pass it to the aforementioned method to start the capture. The result is a simple text field in the live capture creation dialog that allows the user to freely configure a *pcap-filter* to exactly fit their needs as they would using command-line tools like `tcpdump`. This improvement is especially useful if a user needs to analyze traffic incoming from a specific source, for example if there are suspicions of a sinkholed infection under a specific address in the network.

Chapter 6

Evaluation

To test the quality and effectiveness of the features implemented in this work, this chapter evaluates them in the following three ways: First, the features implemented in the previous chapters are evaluated on their performance against a regular real-world DNS server. Second, the effectiveness in mitigating various malware families is tested. Lastly, two case studies investigate how this work's malicious traffic detection and diversion solution could be used in real-world scenarios.

6.1 Performance

While theoretical performance optimizations had been made during implementation of the DNS module, real-world performance remained unknown. In order to determine the latter, the following test was performed. First, a randomly generated blacklist of predetermined length was applied with every entry consisting of 10 random alphanumeric characters with a .com ending (e.g. `xVMMJcf9WF.com`). Second, using a shell-script and the command-line tool `dig`, 100 DNS requests for a domain not in the blacklist were sent to a reference DNS server (e.g. Google's 8.8.8.8) and to the DNS sinkhole running locally, which in turn uses the same server as *mainDNS*. A non-blacklisted domain was chosen for the reason that it represents the worst-case scenario from a performance standpoint: The requested domain is checked with the blacklist, then needs to be requested from *mainDNS* and the resulting response needs to be looked up in the blacklist again before being returned to the requesting client. Thus, testing for this scenario will yield the worst delays possible. The times it took both servers to respond was saved for each iteration and this process was repeated for the blacklist sizes of 10, 100, 1 thousand, 10 thousand and 1 million entries. The results are shown in Figure 6.1 and are rather positive: with the largest tested blacklist of 1 million entries, a relayed request only resolved with an average added response time (delay) of 2.3ms. With these numbers it would be possible to deploy this system to a real-world environment without users noticing any delay compared to normal DNS requests they would perform. Additionally, a high number of requests could be handled, allowing the system to be deployed in an SME network.

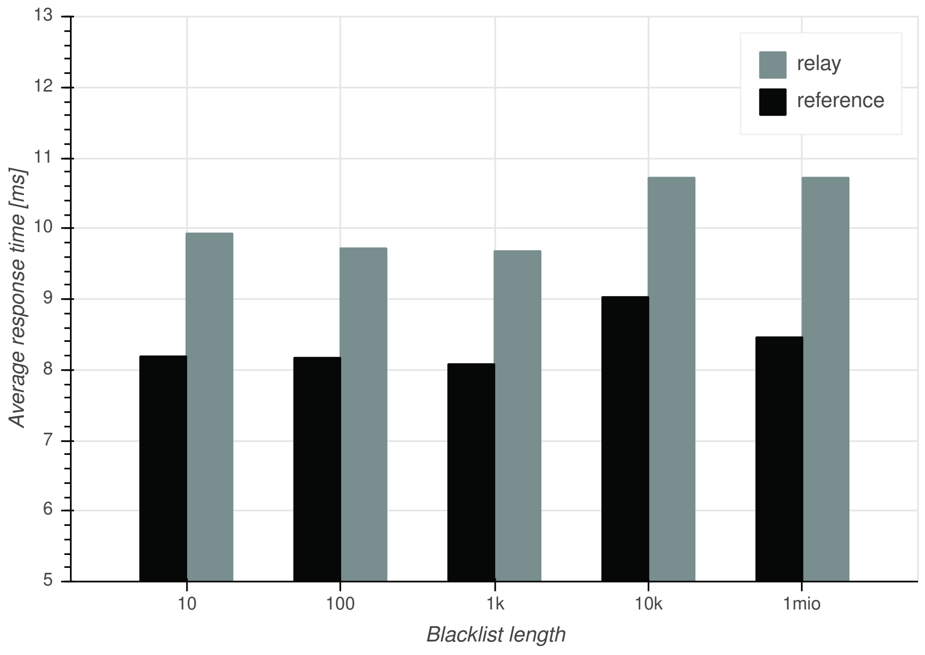


Figure 6.1: The real-world performance achieved by the DNS module. *reference* shows the response time of 8.8.8.8, while *relay* refers to the response time of the DNS module with it's *mainDNS* set to the same server.

6.2 Malware Mitigation

To test the effectiveness of the chosen detection and diversion approach, the behaviour of a selection of real-world malware was tested in a network configured to use the DNS sinkhole proposed in this paper. Before testing began, a fresh Windows 11 virtual machine (*VM A*) was set up from [30] and all antivirus software on it was disabled, including Windows Defender's real-time protection. *VM A* was then placed in a specially created LAN network with a separate virtual machine (*VM B*) already running SecGrid with the DNS and Live Capture modules enabled. On *VM A*, the DNS settings were then set to exclusively use the sinkhole address for all lookups. After this setup process, a snapshot of this VM was created, which was then used as the starting point for every test conducted. For each test, the corresponding sample was directly downloaded onto the virtual machine and executed without higher privileges. All of these samples were chosen from [32] and [45] such that each malware family was represented at least twice.

The aforementioned samples' testing was then done in the following fashion: First, each sample was executed in a control run with an empty blacklist, recording all domains that were contacted during execution. A second run was subsequently made with all previously recorded domains being blacklisted, with SecGrid's newly added filtered live capture feature being used on *VM B* to record all traffic incoming from *VM A*'s IP address only. The testing environment was kept open for 10 minutes in each iteration of testing to allow for as complete a picture as possible.

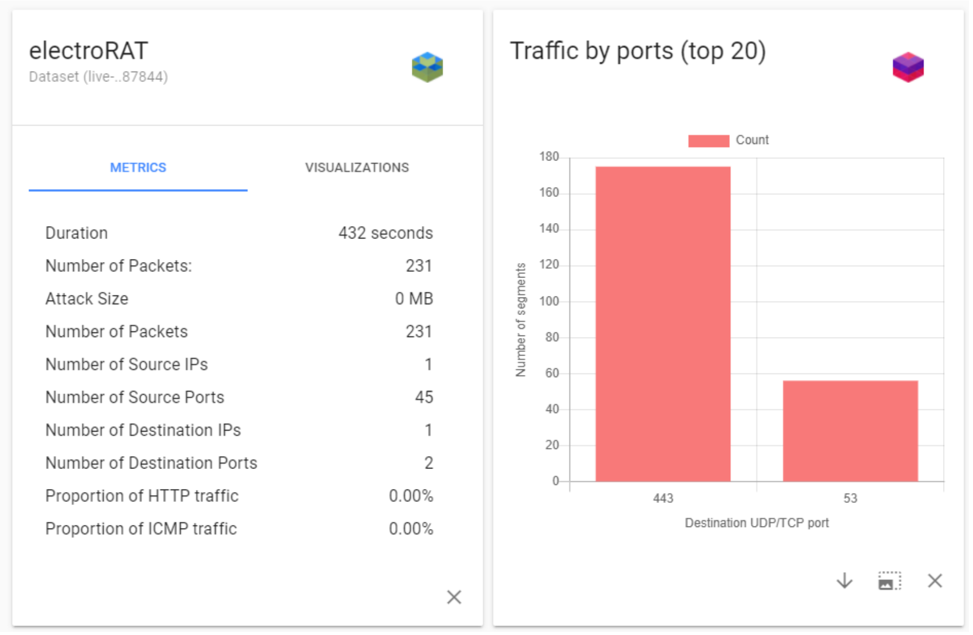
A general overview of the tested malware and the results is shown in Table 6.1. In this

Malware Family	**Instances Tested**	**Result**	**Notes**
Worm	Allaple	Not Mitigated	Does not use DNS, attacks hardcoded targets
Worm	Adylkuzz	Mitigated	Blocking crypto-pool domains mitigated malicious activity
Spyware / Trojan	GravityRAT	Mitigated	Blocking C&C domains mitigates malicious activity
Spyware / Trojan	ElectroRAT	Mitigated	Blocking C&C domains mitigates malicious activity
Ransomware	WannaCry	Mitigated	Killswitch domain mitigates malicious activity
Ransomware	RAASNet	Not Mitigated	Encryption takes place before any network activity
Ransomware	Ransom0	Not Mitigated	Encryption takes place before any network activity
Botnet	RBot	Mitigated	Blocking C&C domains mitigates malicious activity
Botnet	SdBot	Mitigated	Blocking C&C domains mitigates malicious activity
Botnet	EternalRocks	Mitigated	Blocking C&C domains mitigates malicious activity

Table 6.1: Overview of tested malware and the sinkhole's mitigation effectiveness on them.

table, a malware was counted as mitigated when the sinkholing lead to the malware no longer fully fulfilling its malicious intent, for example if a ransomware would no longer encrypt the victim system.

Generally, the sinkhole was very successful in mitigating malware that relied on network communication to receive commands such as botnets and spyware. Here it can generally be said that if the command & control server of the malware was blacklisted, the malware was usually successfully mitigated. This was the case with all tested spyware, which simply idled and periodically tried reconnecting when sinkholed, as shown for the case of ElectroRAT in Figure 6.2. Here it should be noted that both GravityRAT and ElectroRAT still moved their executables into directories hidden from plain sight of the user before attempting any connection and added themselves to the autostart applications on the victim machine. Similar results were achieved with the RBot and SdBot botnets: If their C&C domains were sinkholed, both did not execute any actions besides periodically trying to reconnect using a list of pre-configured target domains. However, this did not apply to the EternalRocks botnet: the latter relies on the Tor project [23] for communication, which it tries to download on startup. If the domain *torproject.org* was sinkholed, no


electroRAT
Dataset (live-..87844)

METRICS VISUALIZATIONS

Duration	432 seconds
Number of Packets:	231
Attack Size	0 MB
Number of Packets	231
Number of Source IPs	1
Number of Source Ports	45
Number of Destination IPs	1
Number of Destination Ports	2
Proportion of HTTP traffic	0.00%
Proportion of ICMP traffic	0.00%

Figure 6.2: A SecGrid visual representation of captured traffic from a sinkholed Electro-RAT test run. It is clearly visible that the malware regularly tried reconnecting to the HTTPS port 443, totaling 175 times over the span of the 432 captured seconds.

further actions would be taken by the malware.

The tested instances of worms yielded mixed results. The first of the tested worms, *Allaple*, did not rely on any DNS at all and instead spreads over the local network using various exploits such as the Windows SMB vulnerability dubbed *EternalBlue* [31] while targeting hardcoded IP addresses with denial-of-service attacks. The second tested worm, *Adylkuzz*, also spreads over the local network, but contrary to *Allaple*, its purpose is not running DoS attacks, instead it attempts using the victim's computational resources to mine cryptocurrency. In order to achieve this, it must contact cryptocurrency-mining related domains, this being *xmr.crypto-pool.fr* in the tested case. Sinkholing this domain effectively blocked the worm from doing anything except for trying to spread itself further to other machines on the local network.

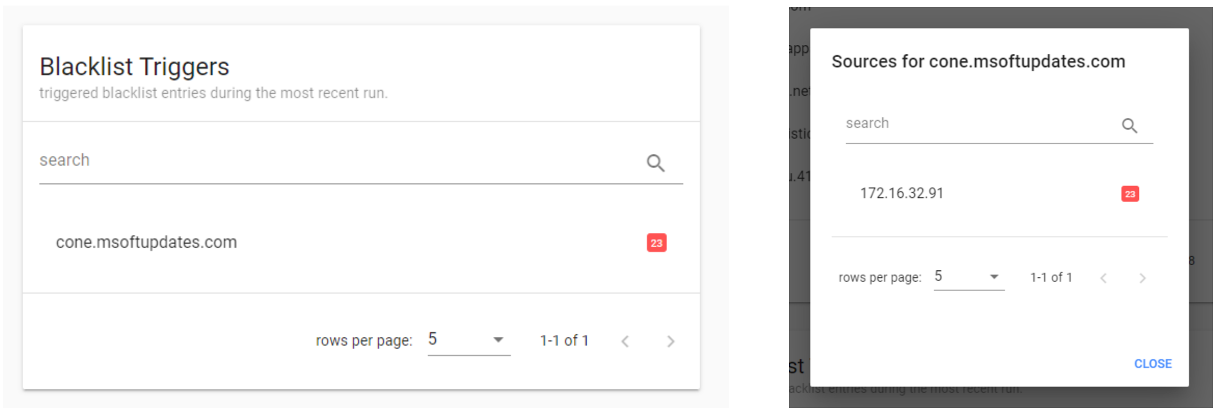
The worst results were achieved in the mitigation of ransomware. Most tested products start encryption immediately after completing infection of the target machine, not relying on any kind of communication or confirmation beforehand. Both RAASNet and Ransom0 start encryption of user data before any other actions that rely on network communication are taken. In addition to observing this behaviour, its intention and correctness can be confirmed in the source code since both RAASNet and Ransom0 are open-source ransomware [44] [28]. In all tested ransomware, the only exception to the aforementioned behavior was presented by WannaCry. This ransomware famously includes a kill-switch in the form of a special domain, which prevents it from taking any further action. While the kill-switch domain for this specific malware is globally registered which therefore makes WannaCry ineffective in real-world use, it should still be noted that sinkholing kill-switch domains will lead to malware being made ineffective and should thus be counted as being mitigated.

6.3 Case Studies

In this section, multiple scenarios will be constructed to better demonstrate the real-world use and effectiveness of this paper's contribution to the existing SecGrid architecture. The first scenario shows a typical use-case where a company uses the sinkhole to identify and prevent the further spread of a threat emerging in their network with the help of an external blacklist. The second scenario describes how the sinkhole can be used in an unexpected way to prevent phishing attacks targeting employees of a company in a country under attack.

6.3.1 Malware Attack

Let us consider an SME (small to medium-sized enterprise) *Alice Corp.* which aims to improve its operational security inside the company network. Since *Alice Corp.* does not possess enough resources to invest in solutions such as SDN-based security systems, it opts to employ a DNS sinkhole instead. *Bob Corp.*, another close-by SME with more resources is running an instance of the SecGrid sinkhole, and maintains its own public blacklist based on open-source lists as well as its own findings inside the network. *Alice Corp.* also decides to deploy the same system, but opts for using *Bob Corp.*'s public blacklist. The corporate network is subsequently configured to exclusively allow DNS requests through the SecGrid sinkhole and using the DNS module's automatic blacklist pulling functionality, *Bob Corp.*'s public blacklist is set to be used and updated hourly.


Blacklist Triggers
triggered blacklist entries during the most recent run.

cone.msoftupdates.com 23

rows per page: 5 1-1 of 1

Sources for cone.msoftupdates.com

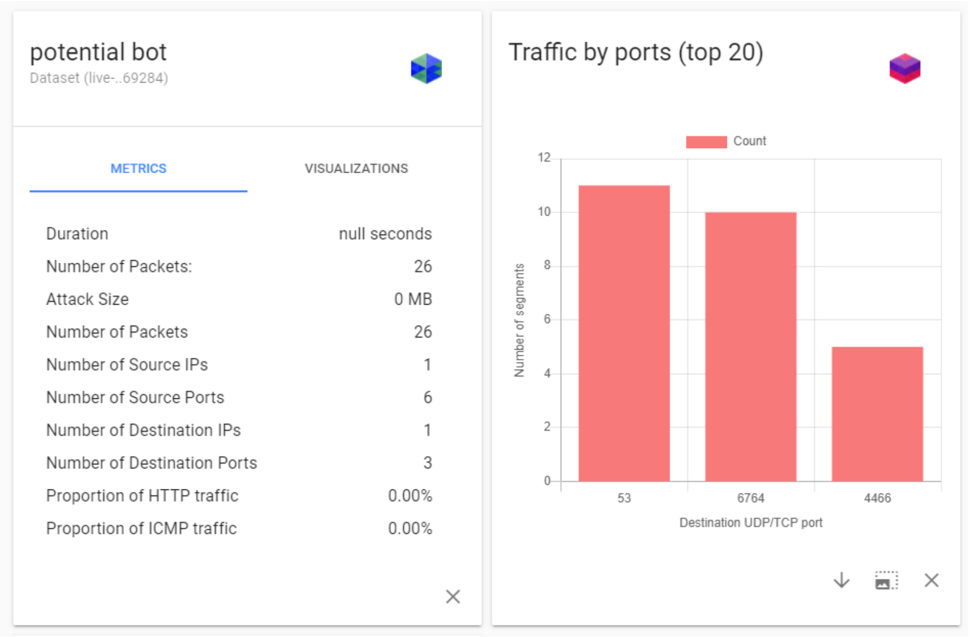
172.16.32.91 23

rows per page: 5 1-1 of 1

Figure 6.3: The statistics card of the ficticious *Alice Corp.* shows one IP address regularly trying to access a known malicious domain, prompting suspicion.

After the system has been established, some time goes by uneventfully. However, soon *Bob Corp.*'s security team detects a new botnet variation spreading through their networks. After analyzing samples they collected from infected machines, they isolate the domain *cone.msoftupdates.com* as the main Command & Control endpoint. Shortly, it is added to their publicly available blacklist. *Alice Corp.*'s SecGrid instance then updates the blacklist within one hour to include the latest malicious domain, without requiring any action.

The next day, *Alice Corp.*'s technical team will notice a high number of DNS requests coming through for this newly blacklisted domain from a specific IP address within their network, which would look similar to Figure 6.3. This will prompt suspicion, and warrant a capture of incoming traffic from this address. Using SecGrid's newly integrated live capturing capabilities, the team starts a live capture on the relevant interface and uses the filtering function to only capture incoming traffic from the suspicious source IP using a filter such as `src 172.16.32.91`.



potential bot
Dataset (live-..69284)

Duration	null seconds
Number of Packets:	26
Attack Size	0 MB
Number of Packets	26
Number of Source IPs	1
Number of Source Ports	6
Number of Destination IPs	1
Number of Destination Ports	3
Proportion of HTTP traffic	0.00%
Proportion of ICMP traffic	0.00%

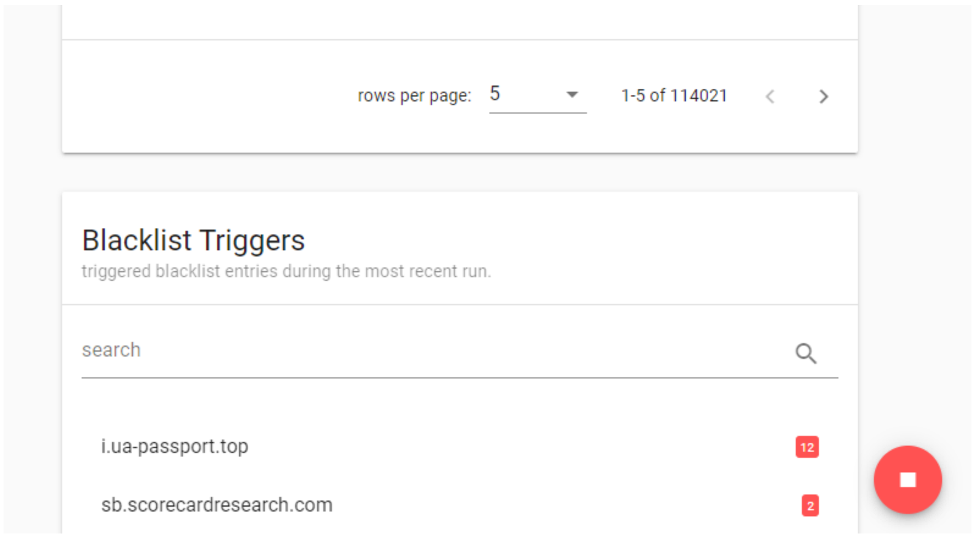
Figure 6.4: A SecGrid visualization of a live capture of a malware infected host.

When the live capture concludes, the team uses SecGrid to visualize the data (as shown in Figure 6.4 and finds 15 connection attempts on two ports not assigned to any usual services, confirming the suspicion. After collecting the infected machine from its user, a further digital forensics investigation shows an infection with the same malware as *Bob Corp.* recently identified. Thus, the malware can safely be removed from the machine and prevented from spreading any further through the network.

This case study shows that the new DNS sinkhole capabilities implemented in SecGrid can help prevent possible cyberattacks while also allowing for easy detection of possible breaches and infections without heavy investments into high-tech networking infrastructure or software. Additionally, shared or centralised lists can be used in order to lower the load on technical staff while keeping up to date with the newest threats.

6.3.2 Phishing Campaigns

In this scenario, we will consider *Carol Corp.*, an SME focusing on producing local print media. This corporation does not focus on or put considerable resources into information security and security training of their employees, but is using SecGrid's DNS sinkhole feature as an easy preventative measure for a surface-level security improvement. *Carol Corp.*'s configuration of the SecGrid sinkhole uses a publicly available blacklist which is partly crowd-sourced, partly maintained by a security researcher and is updated every hour to include the latest threats. The deployed SecGrid configuration then polls a current version of this list at every hour from a publicly exposed URL.


rows per page: 5 1-5 of 114021

Blacklist Triggers
triggered blacklist entries during the most recent run.

search

i.ua-passport.top 12
sb.scorecardresearch.com 2

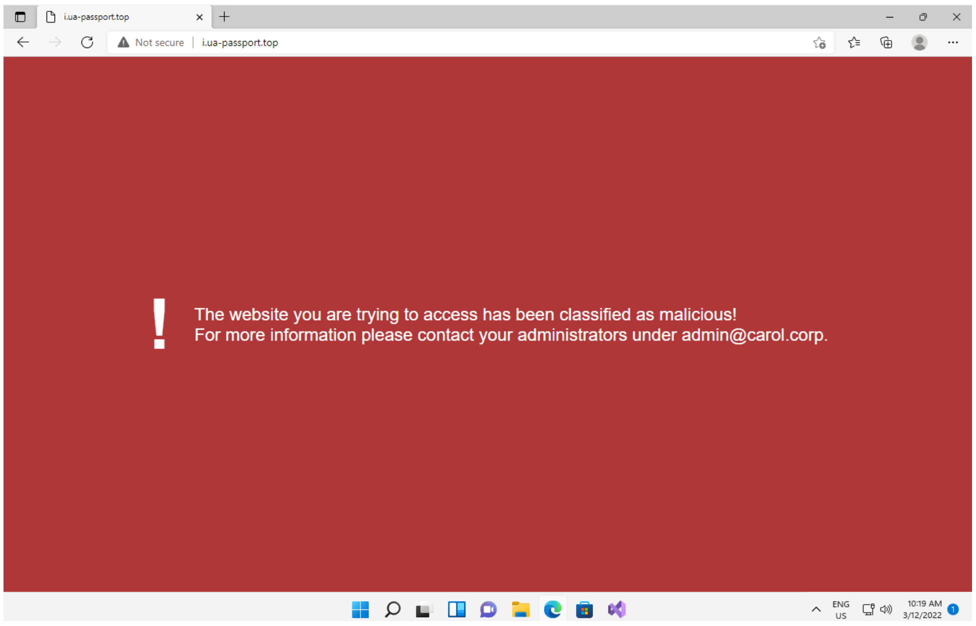
Figure 6.5: *Carol Corp.*'s dashboard showing a suspicious entry.

Due to external circumstances, the country *Carol Corp.* is located in finds itself under an increased amount of cyber-attacks recently, most of which are carried out by nation-state adversaries. Many of these attacks take the form of phishing campaigns targeted towards media companies in order to gain control of local information spread. Since these attacks often target a large amount of victims, information on these attacks quickly spreads and users of the public blacklist quickly identify and collect domains connected to phishing attempts observed around the country. Soon these domains are pulled into the local blacklist of *Carol Corp.*'s SecGrid instance through the continuous URL polling feature. During a routine dashboard inspection a few days later, admins of *Carol Corp.* find suspicious entries in their statistics card, as shown in Figure 6.5.

After inspecting the domains in question, they are identified as malicious phishing-related domains originating from the same nation-state adversary as many of the recent attacks against other media companies around the country. Quickly, *Carol Corp.* decides to implement measures for mitigation. First, using the sources function of the statistics card, the user that was targeted in the attack is identified and an investigation into the potential damage achieved by the attack is launched. To prevent damage from unaware users, preventative measures are put in place: An http server is installed on the sinkhole machine, showing a warning page (as shown in Figure 6.6) for potentially visiting users, deterring them from engaging any further. At the same time, the company launches an awareness campaign to decrease the probability of users even engaging with potential phishing.

Using this strategy in combination with SecGrid's DNS sinkholing capabilities, *Carol Corp.* is able to successfully withstand the phishing attacks in the following months and prevent any larger damage from occurring to the corporation.

Even though this work's initial goals are centered around preventing cyberattacks originating from malware, this case study shows that SecGrid's new sinkholing capabilities



The website you are trying to access has been classified as malicious!
For more information please contact your administrators under admin@carol.corp.

Figure 6.6: *Carol Corp.*'s warning site that is shown when trying to visit a blacklisted domain.

can also be applied outside of this scope to prevent other kinds of cyberattacks such as phishing successfully as well.

Chapter 7

Summary, Conclusions and Future Work

This thesis was mainly concerned with the design and implementation of a network traffic sinkhole integrated into the *SecGrid* platform to allow for easy diversion and subsequent analysis of malicious traffic. After introducing the reader to the motivation and description of this work in the first chapter, the necessary background and related works such as alternative solution approaches were laid out in the following two chapters. Using the knowledge gained in these chapters, DNS in conjunction with a blacklist is chosen as the traffic detection and diversion approach for its low intrusiveness and easy deployability. The following chapter details the necessary changes in the existing *SecGrid* architecture, namely the addition of two new modules: the *live capture* module and the *DNS module*, as well as corresponding additions in the user interface. The implementation of these two modules and changes are documented in the next chapter. The completed implementation features a full-fledged DNS-sinkhole with a configurable blacklist, either manually by sending entries directly, or via URL, which can be used once or set to be polled in regular intervals. A new part in *SecGrid*'s user interface allows control of all of the aforementioned settings as well as monitoring and controlling the state of the sinkhole itself. While the sinkhole is running, operators also have access to an overview that shows the most requested blacklisted domains during the current run and the sources of these requests.

As is shown in the evaluation chapter of this work, the software was first tested for performance. These tests could be concluded with positive results: The relaying of the requests and checking in the blacklist introduced an additional delay of only 2.3ms. This low overhead would allow for deployment of the system without users noticing any delay in real-world networks. The sinkhole was then tested for each malware type previously established with varying results: While the sinkhole was less effective in mitigating ransomware and worm attacks since these malware types relied on hard-coded behaviour and target addresses, tests with real-world samples of spyware, trojans and botnets showed that malware relying on DNS-based C&C servers could not only be prevented from fulfilling its malicious intent, but the attempted malicious connections could also successfully be recorded for further analysis with the direct capture module. It can thus be concluded that the initial goal of integrating easily configurable DNS sinkhole and direct capture capabilities was successfully reached, with generally positive effectiveness results being achieved on relevant malware samples.

7.1 Limitations and Future Work

Though this work's goals have been reached, some limitations arise from the nature of the chosen approach. One such limitation arises from relying purely on DNS and blacklists for malware detection. This approach limits this work to detecting previously known threats only. Additionally, manually detecting suspicious behaviour without the help of a blacklist is not possible using the current interface. Lastly, if threats are detected, users of the system are not notified and need to regularly navigate to the statistics card in order to stay up to date.

To address these shortcomings, future work could explore additions to the system and interface that were outside of the scope set for this paper. One such addition could be the sending of alerts to users of SecGrid: instead of only displaying triggers of the blacklist, such breaches could also be sent via e-mail or other channels to responsible users.

Another addition could be an improved statistics card. Instead of only showing which source triggered which address, more comprehensive statistics of DNS requests could be gathered and composed. Such statistics could include: most active sources, most requested domains (even non-blacklisted ones), request frequency over time and more. Such statistics could provide a deeper insight into the network's activity and could help administrators spot potential undetected threats faster (e.g. if an infected machine repeatedly and consistently requests the same C&C domain).

Using these statistics, suspicious activity could also potentially be automatically detected, perhaps leveraging modern machine-learning technology for pattern-recognition. Detected suspicious activity could then be highlighted in the user-interface, or on sufficient confidence even be sent via the aforementioned alert channels.

Bibliography

[1] Abuse.ch. *URLhaus*. Accessed 14.01.2022. URL: `https://urlhaus.abuse.ch/`.

[2] D Arivudainambi, Varun Kumar KA, P Visu, et al. "Malware traffic classification using principal component analysis and artificial neural network for extreme surveillance". In: *Computer Communications* 147 (2019), pp. 50–57.

[3] Davide Baglieri. *DigitalSide Threat-Intel*. Accessed 14.01.2022. URL: `https://osint.digitalside.it/Threat-Intel/lists/latestdomains.txt`.

[4] Robert Barrett et al. "Dynamic traffic diversion in SDN: testbed vs mininet". In: *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2017, pp. 167–171.

[5] Leyla Bilge et al. "Exposure: A passive dns analysis service to detect and report malicious domains". In: *ACM Transactions on Information and System Security (TISSEC)* 16.4 (2014), pp. 1–28.

[6] Steven Black. *Unified hosts file with base extensions*. Accessed 01.02.2022. URL: `https://github.com/StevenBlack/hosts`.

[7] Luc Boillat et al. "A Tool for Visualization and Analysis of Distributed Denial-of-Service (DDoS) Attacks". In: *Communication Systems Group, Department of Informatics, Universität Zürich* (2020).

[8] Wouter Borremans and Ruben Valke. "BGP (D)DoS Diversion". In: (2005).

[9] Guy Bruneau and Rick Wanner. "DNS Sinkhole". In: *SANS Institute InfoSec Reading Room, Aug* 7 (2010).

[10] Kevin Butler et al. "A survey of BGP security issues and solutions". In: *Proceedings of the IEEE* 98.1 (2009), pp. 100–122.

[11] RIPE Network Coordination Centre. *What is an AS Number?* Accessed 16.01.2022. Nov. 2019. URL: `https://www.ripe.net/manage-ips-and-asns/as-numbers`.

[12] Jenny Costa et al. "IoT-Botnet Detection using Long Short-Term Memory Recurrent Neural Network". In: *International Journal of Engineering Research & Technology* 9.8 (2020).

[13] Manuel Egele et al. "Dynamic spyware analysis". In: (2007).

[14] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. "A survey of botnet and botnet detection". In: *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. IEEE. 2009, pp. 268–273.

[15] FireHOL. *firehol_level1*. Accessed 14.01.2022. URL: `https://iplists.firehol.org/?ipset=firehol_level1`.

[16] OpenJS Foundation. *Node.js*. URL: `https://nodejs.org/en/`.

[17] Muriel Franco et al. “SecGrid: a Visual System for the Analysis and ML-based Classification of Cyberattack Traffic”. In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE. 2021, pp. 140–147.

[18] David Freet and Rajeev Agrawal. *A Statistical Comparison of Security Visualization Efficiency Compared to Manual Analysis of IDS Log Data*. IEEE, 2018.

[19] Keving Gennuso. *Shedding light on security incidents using network flows*. 2012.

[20] The Tcpdump Group. *pcap-filter(7) man page*. Accessed 03.02.2022. URL: `https://www.tcpdump.org/manpages/pcap-filter.7.html`.

[21] Nico Hinze et al. “On the potential of BGP flowspec for DDoS mitigation at two sources: ISP and IXP”. In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. 2018, pp. 57–59.

[22] Nicholas Ianelli and Aaron Hackworth. “Botnets as a vehicle for online crime”. In: *CERT Coordination Center* 1.1 (2005), p. 28.

[23] The Tor Project Inc. *The Tor Project*. Accessed 23.02.2022. URL: `https://www.torproject.org/`.

[24] Ismahani Ismail, Sulaiman Mohd Nor, and Muhammad Nadzir Marsono. “Stateless malware packet detection by incorporating naive bayes with known malware signatures”. In: *Applied Computational Intelligence and Soft Computing* 2014 (2014).

[25] Hyun Mi Jung, Haeng Gon Lee, and Jang Won Choi. “Efficient malicious packet capture through advanced DNS sinkhole”. In: *Wireless Personal Communications* 93.1 (2017), pp. 21–34.

[26] Martin Karresand. “Separating Trojan horses, viruses, and worms-a proposed taxonomy of software weapons”. In: *IEEE Systems, Man and Cybernetics SocietyInformation Assurance Workshop, 2003*. IEEE. 2003, pp. 127–134.

[27] Manjeri N Kondalwar and CJ Shelke. “Remote Administrative Trojan/Tool (RAT)”. In: *Int. J. Comput. Sci. Mob. Comput* 3333.3 (2014), pp. 482–487.

[28] Hugo Lebelzic. *Ransom0*. Accessed 21.03.2022. URL: `https://github.com/HugoLB0/Ransom0`.

[29] Gonzalo Marín, Pedro Caasas, and Germán Capdehourat. “Deepmal-deep learning models for malware traffic detection and classification”. In: *Data Science–Analytics and Applications*. Springer, 2021, pp. 105–112.

[30] Microsoft. *Get a Windows 11 development environment*. Accessed 23.02.2022. URL: `https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/`.

[31] Microsoft. *Microsoft Security Bulletin MS17-010*. Accessed 21.03.2022. URL: `https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010`.

[32] Fabrizio Monaco. *malware-samples*. Accessed 23.02.2022. URL: `https://github.com/fabrimagic72/malware-samples`.

[33] *node-pcap*. Accessed 03.02.2022. URL: `https://github.com/node-pcap/node_pcap`.

[34] Harun Oz et al. “A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions”. In: *arXiv preprint arXiv:2102.06249* (2021).

[35] *Pi-hole®*. Accessed 01.02.2022. URL: `https://pi-hole.net/`.

[36] Carlo Pugnetti and Carlos Casián. “Cyberrisiken und Schweizer KMU: eine Untersuchung der Einstellungen von Mitarbeitenden und verhaltensbedingter Anfälligkeiten”. In: (2021).

[37] Rishikesh Sahay et al. "Cybership: An SDN-based autonomic attack mitigation framework for ship systems". In: *International Conference on Science of Cyber Security*. Springer. 2018, pp. 191–198.

[38] Praghat Kumar Singh. "A physiological decomposition of virus and worm programs". PhD thesis. Citeseer, 2002.

[39] Snort. *Snort IP Block List*. Accessed 14.01.2022. URL: `https://snort.org/downloads/ip-block-list`.

[40] Liu Song. *dns2*. URL: `https://www.npmjs.com/package/dns2`.

[41] Zheng-Zhi Tang et al. "Malware Traffic Classification Based on Recurrence Quantification Analysis." In: *Int. J. Netw. Secur.* 22.3 (2020), pp. 449–459.

[42] Swiss Government Computer Emergency Response Team. *Vulnerable Systems Timeline*. Accessed 28.02.2022. URL: `https://www.govcert.admin.ch/statistics/timeline/`.

[43] D Turk. "Configuring BGP to block Denial-of-Service attacks". In: *RFC 3882* (2004).

[44] Leon Voerman. *RAASNet*. Accessed 21.03.2022. URL: `https://github.com/leonv024/RAASNet`.

[45] ytisf. *theZoo - A Live Malware Repository*. Accessed 23.02.2022. URL: `https://github.com/ytisf/theZoo`.

[46] Saman Taghavi Zargar, James Joshi, and David Tipper. "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks". In: *IEEE communications surveys & tutorials* 15.4 (2013), pp. 2046–2069.

Abbreviations

AS	Autonomous Systems
BGP	Border Gateway Protocol
C&C	Command & Control
CORS	Cross-Origin Resource Sharing
DDoS	Distributed Denial of Service
DNS	Domain Name System
DPI	Deep Packet Inspection
ES6	EcmaScript 6
GovCERT	Computer Emergency Response Team of the Swiss Government
IT	Information Technology
IP	Internet Protocol
JSON	JavaScript Object Notation
PoC	Proof of Concept
RAT	Remote Administration Tool
REST	Representational State Transfer
SDN	Software Defined Networks
SME	Small to Medium-Sized Enterprise
SSRF	Server-Side Request Forgery
VM	Virtual Machine
UI	User Interface

Glossary

Authentication describes the process of verifying an entity has the identity that it claims it has.

Authorization is the decision whether an entity is allowed to perform a particular action or not, e.g. whether a user is allowed to attach to a network or not.

Autonomous Systems are large computer networks or even groups of networks that share a single routing policy. Usually AS are assigned a unique number by a local authority, which is then called their Autonomous System Number (ASN).

Botmaster refers to an individual in control of a number of infected machines, able to control them and issue commands of many kinds.

Diversion in the context of this work describes to the act of redirecting traffic away from it's originally intended destination to an alternative one.

polling is the act of asking a server for certain information.

pulling describes the act of using a certain amount of force in order to move data towards the client.

List of Figures

List of Tables

Appendix A

Installation Guidelines

Since this work focused on adding functionality to the existing SecGrid platform and no dependencies requiring special care have been added, the installation instructions remain identical to those of SecGrid. The installation instructions presented in this section assume the user intends on using the default DDosGrid-provided OAuth authorization scheme. For detailed instructions on using third-party providers see Appendix A in [17].

A.1 Development Environment

As a prerequisite for all the components, the user needs to have the repository that home the three major components. This can be done using the contents of the CD or by cloning from the SecGrid GitHub repository and checking out the newly added branch `sinkhole-integration` which contains this paper's work:

```
git clone https://github.com/ddosgrid/ddosgrid-v2.git
cd ddosgrid-v2
git checkout sinkhole-integration
```

Additionally, the user needs to have the libraries and tools installed for which we refer the user to the official documentation:

1. Node.js
2. npm
3. git
4. SSH agent
5. Python 3.6 or higher
6. libpcap

A.1.1 Miner

Even though no major changes have been made to the *miner* sub-project, the necessary dependencies still need to be installed in order for the backend to work properly. This installation can easily be done using the following commands:

```
cd miner
npm i
```

A.1.2 Backend (API)

The backend houses the DNS and live capture module introduced with this work. Before starting the backend, some dependencies must be installed using the following commands:

```
cd api
npm i
```

In order to run the backend, it is necessary to be situated in the `./api` folder of the project, and run the `start_dev_server.sh` script without changing the working directory:

```
cd api
sudo scripts/start_dev_server.sh
```

Note that if intentions exist to run the DNS sinkhole on the default DNS port 53, the backend must be started with root permissions, thus the usage of `sudo` in the above instructions.

A.1.3 Frontend

The last part of the application is the SecGrid frontend. For that, we navigate to *./frontend* and fetch the dependencies using `npm i`. After that, we can run the development web server using `npm run serve` which brings up a server that automatically restarts when the codebase changes:

```
 DONE  Compiled successfully in 6698ms


  App running at:
  - Local:   http://localhost:8081/ddosgrid/
  - Network: http://192.168.13.46:8081/ddosgrid/
```

As we can see, the server is listening on port 8081.

Finally, for the full application to work properly, both the backend and the frontend should be running at the same time.

Appendix B

Contents of the CD

The CD supplied with this paper contains a number of important resources tied to its contents. The following folder structure can be found on it:

```
evaluation/
    performance/
paper/
project/
    ddosgrid-v2/
slides/
    midterm/
```

`evaluation/performance/` contains all materials used to execute the performance tests described in Section 6.1. This includes the script used to generate the blacklists of all sizes (`generate-blacklist.js`), as well as the shell script used to evaluate them (`dns-benchmark.sh`). Additionally, all raw data generated from these tests is included in the files of the naming format `results-<BLACKLIST_SIZE>.csv`. Lastly, the python jupyter notebook used to generate the graphs and reported results is also included in this directory (`bsc-evaluation.ipynb`).
For security reasons, the malware samples used in mitigation testing are not included on this CD, however they can easily be downloaded from the source GitHub repositories ([32] and [45]).

`paper/` is the directory where all LaTeXsource files used for this paper have been deposited.

`project/ddosgrid-v2/` contains all relevant source code used for this work. Alternatively, these same contents can be cloned from GitHub using the instructions seen in Appendix A.

`slides/midterm/` accomodates both the source and PDF files of the slides used in the midterm presentation of this work.