Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

Detection of Amplifiers using Active Measurements

Hamza Zafar

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

Detection of Amplifiers using Active Measurements

Erkennung von Amplifiern mit aktiven Messungen

Author:	Hamza Zafar
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Simon Bauer, M. Sc. Oliver Gasser, M. Sc. Dipl.-Inform. Stefan Metzger
Date:	March 21, 2019

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, March 21, 2019	
Location, Date	Signature

Abstract

Bandwidth amplification attacks are a special type of DDoS attacks in which the response packet is significantly larger than the request packet. An attacker can exploit the disparity in request and response sizes and use the source IP address spoofing to backscatter large amounts of traffic to exhaust the victim's bandwidth. We developed a framework for detecting amplifiers using active network measurements. The framework is capable of performing Internet-wide scans. We evaluated our framework by conducting scans in Munich Scientific Network for detection for ten UDP-based protocols which can be abused for amplification. We followed the practices to conform to the ethical considerations of conducting network scans. We analyzed the scan results to determine the characteristics of amplifiers and uncovered the misconfigurations which can expose the devices to amplification abuse. We described the strategies which can be employed to proactively mitigate the amplification threat.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Denial of service attacks are aimed at making the services unavailable for legitimate users. Countermeasures are being developed constantly by security researchers to mitigate the denial of service attacks, but the attackers are continuously searching for and exploiting the vulnerabilities in widely deployed protocols. The attackers target to consume the compute, storage and network-based resources of the victim.

One such denial of service attack is based on consuming the victim's bandwidth. For this purpose, the attacker sends requests with victim's spoofed source address to services which in response send large responses to the victim. Making use of several devices to send spoofed requests to the victim can exhaust its bandwidth. The disparity in the request and response sizes and the lack of authentication in services can be exploited to generate several Gbps of network traffic towards the victim by using only a fraction of the generated bandwidth. This type of attack is known as a Distributed Reflective Denial of Service Attack (DRDoS).

Many internet companies report severe DRDoS attacks, it has been observed that the attacks are getting worse with time. In 2013, a spam activities tracking organization named Spamhaus [42] observed a DNS amplification attack peaking at 300 Gbps. In 2018, GitHub [19] faced an attack that reached 1.35 Tbps, the attack involved a misconfiguration in Memcached servers which were publicly available on the internet. Arbor Networks also reported a similar DRDoS attack in which Memcached servers were used. They reported a peak of 1.7 Tbps bandwidth [31], which is the highest attack bandwidth reported to date.

1.1 Motivation

With the popularity of Internet of Things (IoT) a large number of smart devices are getting connected to the internet. The IoT devices are criticized for the lack of security considerations in their design. If an attacker gains control of an IoT device, it is not only a threat to the user's privacy but it could also be used by the attackers to launch large scale DoS attacks. In 2016, a DNS provider named DYN faced an attack [13] which employed the IoT devices e.g. printers, IP cameras, and smart TVs infected with the Mirai malware. The Mirai malware scanned the internet for publicly available IoT devices and used the factory default username and passwords to gain control of the devices. Later, the devices were used to attack the internet's DNS infrastructure, which rendered many websites inaccessible to the users.

The first step for the attackers in launching a DRDoS attack is to search the internet for devices which could be used to generate network traffic to consume the victim's bandwidth, such devices are also known as amplifiers. The amplifiers are either misconfigured or they run a vulnerable service which should only be used by the hosts in a local network. Amplifiers should be detected by the network administrators and proper steps should be taken against them to mitigate the amplification threat.

In this context, we have developed a framework which is capable of detecting amplifiers in a network. In addition to scanning an organization's network, the framework is fully capable of executing internet-wide scans. Such internet-wide scans could be conducted for research purposes. The following features are provided by the framework.

- Support for scheduling scanning tasks for detection of amplifiers.
- Support for scanning IPv4 and IPv6 address ranges.
- Fast asynchronous execution of scans using ZMap [12].
- Several dashboards for visualization of scan results.
- Support for e-mail notifications.

1.2 Research Questions

The research questions addressed in this thesis revolve around the detection of amplifiers in a large scale network using active network measurement. We researched the mechanism for scheduling and orchestration of scans for amplifer detection. We highlighted the characteristics of amplifiers such as the change in the amplification factor and the change in the number of active amplifiers over a period of time. We analyzed the detected amplifiers in our network measurements to indicate the causes of amplification vulnerabilities. We also addressed the question of ethical considerations and best practices for scanning networks. Last but not least, we researched the solutions for mitigation of amplification vulnerabilities.

1.3 Outline

This thesis is structured as follow: In Chapter 2, we describe the types of DoS attacks and the infrastructure used by attackers to launch a DRDoS attack. We also discuss the role of source IP address spoofing and connection-less UDP protocol in executing a bandwidth amplification attack. In Chapter 3, we present ten protocols which are frequently exploited for amplification attacks. We briefly introduce each protocol and describe how the attackers exploit them to launch a DRDoS attack. In Chapter 4, we provide details about the features provided by our framework. We also discuss the design and architecture of the framework and the components that make it is scalable and effective to perform an internet-wide scan. In Chapter 5, we describe our measurement setup and the analysis of the scans we have performed using the framework. In Chapter 6, we discuss the related work about amplification attacks and their detection. We also compare our results with the related work. In Chapter 7, we provide details about protecting services against their abuse for amplification attacks, we also discuss the future work and conclude this thesis with a summary.

Chapter 2

Background

2.1 Denial-of-service Attacks

A denial-of-service (DoS) attack is an attack in which an attacker aims to render a computer service inaccessible to its legitimate users. The outcomes of a DoS attack for the victim varies from a minor inconvenience to a major financial loss. The attack is generally operated by oversaturating the capacity of the targeted service until it is not able to process the requests originating from normal users. In a DoS attack, a single source is used to operate the attack. The three major categories of DoS attacks including some examples are discussed below:

2.1.1 Application Layer Attacks

The application layer based DoS attacks also known as layer 7 DoS attacks are generally targeted towards web servers. The goal of the attacker is to exhaust the server's system resources such as storage, memory, and CPU time. The foundation of layer 7 attacks is based on the fact that creating and sending a request is quite inexpensive as compared to the functions executed on the server side to generate a response.

Usually, a server has to perform cache lookups, execute expensive database queries or run some computationally expensive functions to generate a response. The attacker attempts to repeatedly trigger the execution of such resource-consuming functions on the server to render it useless. This significant difference in the amount of work done between a client and a server is exploited by attackers to launch a layer 7 attack. One example of the layer 7 attacks is the HTTP flood attack.

HTTP Flood Attack

HTTP Flood is a type of layer 7 DoS attack in which an attacker exploits the legitimate HTTP requests to bring down a web server. The HTTP Flood attack requires fewer resources to execute and it does not involve any reflection or spoofing techniques. This kind of attack requires very detailed knowledge about the software architecture and design of an application. The attacker repeatedly sends some HTTP requests to trigger complex server-side processing which results in extra resource consumption leading to a denial of service for other users.

HTTP flood attack has two variants, the first one is the HTTP GET flood attack in which an attacker sends HTTP GET requests repeatedly to fetch static content from the web server. The other variant is the HTTP POST flood attack in which an attacker sends HTTP POST requests to create resources on the server, it involves executing validation checks on the request data and running database queries for persisting the data. The attacker exploits the disparity between the resources consumed by an HTTP POST request and its response.

The HTTP flood attack is difficult to detect and mitigate because the requests sent by the attacker are very similar to those of the legitimate users. Generally, rate limiting and throttling based schemes are deployed to defend against the HTTP Flood attacks.

2.1.2 Protocol Attacks

The protocol based DoS attacks exploit the weaknesses in the layer 3 and layer 4 of the OSI model. They are also known as state-exhaustion attacks because they cause the denial-of-service by exhausting the state tables of servers and intermediate networking infrastructure e.g. routers, switches, and load balancers. The exhaustion of resources of the networking equipment results in the denial-of-service for legitimate users. Two popular examples of protocol attacks are *SYN flood* and *Ping of death* attacks.

SYN Flood Attack

The SYN flood attack exploits the TCP's three-way handshake mechanism for connection establishment. The three-way handshake is shown in the Figure 2.1 and the steps it involves are described below:

1. Client sends a SYN packet to request for establishing a TCP connection.
2. Server acknowledges the clients's requests by responding with a SYN-ACK packet.
3. Client replies with an ACK packet to complete the connection establishment.

In a SYN Flood attack, an attacker sends a large number of SYN packets to the target, the target server then responds with a SYN-ACK packets and waits for the final ACK packet from the source. The attacker never replies with an ACK packet which leads to resource consumption on the server side. Since the networks are prone to congestion, the target site waits for some time before removing the half-open connection states from its memory. The SYN flood eventually exhausts the target's resources, thereby not allowing it to establish connections with the legitimate users.

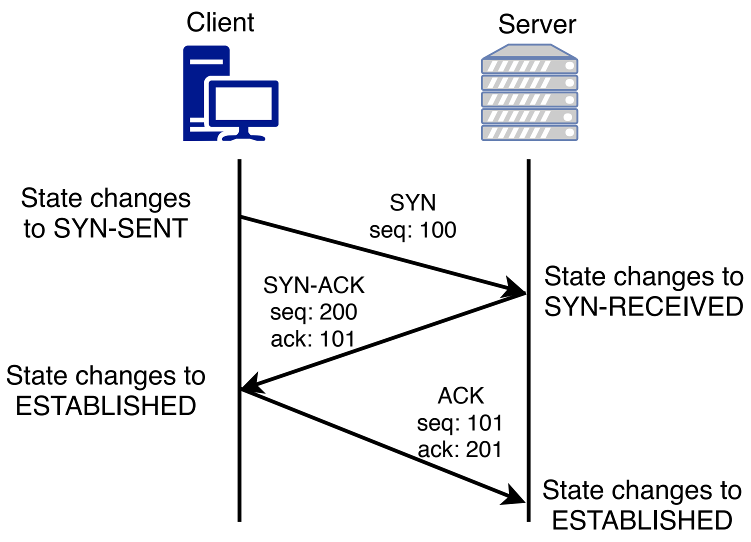

Client
Server
State changes to SYN-SENT
SYN seq: 100
State changes to SYN-RECEIVED
SYN-ACK seq: 200 ack: 101
State changes to ESTABLISHED
ACK seq: 101 ack: 201
State changes to ESTABLISHED

Figure 2.1: TCP's three-way handshake.

Ping of Death Attack

A ping of death attack involves sending a malformed packet to the target computer system which results in its crash. RFC 791 [21] indicates that the maximum size of an IPv4 packet including the IP header is 65535 bytes. Legacy implementations of the networking stack were not designed to handle ping packets larger than the size mentioned in the Internet Protocol. The attackers could exploit this edge case by sending a malformed or oversized packet to the target. During the fragmentation reassembly on the target machine, a buffer overflow can occur, leading to the system crash.

2.1.3 Volumetric Attacks

In a volumetric based DoS attack, an attacker sends a high amount of traffic to a victim to consume its network bandwidth. The consumption of network bandwidth results in preventing the requests from legitimate users to reach the victim services. The attacker exploits the publicly available network services such as DNS, NTP etc. to generate large responses and uses the spoofing technique to direct the responses towards the victim. Such volumetric attacks are also known as *bandwidth amplification attacks*.

The services that can be used as a medium to generate and send high network traffic towards the victim are known as *amplifiers*. The disparity in the size of request and response packets and the lack of authentication mechanism in amplifier services makes this attack relatively simple to execute. The effects of volumetric based attacks are devastating for the victim because using only a few Mbps of bandwidth an attacker can generate a network traffic of several hundred Gbps for the victim. The examples of volumetric based DoS attacks are discussed in detail in the next sections.

2.2 Distributed Denial-of-Service Attack

In case of a DoS attack, a single source machine is used by the attacker. A single attacking source is usually not enough to completely shut down the target services. If a single source is used, it could be easily detected by the system administrators and a firewall could be configured to drop the network packets coming from the source. Moreover, there are limitations for the number of connections a single system could establish or the amount of network bandwidth it could use. These factors makes it less attractive to launch a DoS attack from a single system.

Therefore, in order to create devastating effects on the target services, attackers use multiple systems to consume the bandwidth of the victim. Such a DoS attack that originates from multiple systems is known as a Distributed Denial-of-Service Attack (DDoS). A DDoS attack scales linearly with the number of systems taking part in attacking the victim. A large number of attacking sources in a DDoS attack makes it stealthier and difficult to mitigate as there are chances of unintentionally blocking the requests from legitimate users.

2.2.1 Botnet

Attackers often use botnets to launch a DDoS attack on a large scale. Botnets are a group of devices e.g. smartphones, IoT devices, and personal computers which are connected to the internet and are controlled by an attacker. The individual infected devices are called bots and they have a malware running on them. The attacker sends commands to the bots to operate a DDoS attack. Botnets not only help the attacker to launch a successful DDoS attack but also hides his identity because the requests originate from bots instead of the attacker's own system. The attacker communicates with the botnet in a client-server model, however, some advanced botnet topologies such as star network topology, hierarchical network topology, and peer-to-peer [44] are also being used by the attackers.

2.3 Distributed Reflection Denial-of-Service Attack

Distributed Reflection Denial-of-Service Attack (DRDoS) is a special type of DDoS attack in which an attacker sends requests with spoofed IP address of the victim to publicly available network services to exhaust the victim's bandwidth. The publicly available network services act as amplifiers to generate large responses for a request size of a few bytes. Instead of using personal computing resources, attackers instruct botnets to send spoofed requests to the amplifiers.

Exploiting the disparity in the request and response sizes, attackers are able to launch DRDoS attacks with hundreds of Gbps. This type of attack is also known as bandwidth amplification attack. In the rest of the report, DRDoS attack will be referred as the bandwidth amplification attack. The Figure 2.2 shows a bandwidth amplification attack in which an attacker uses a botnet to attack a victim.

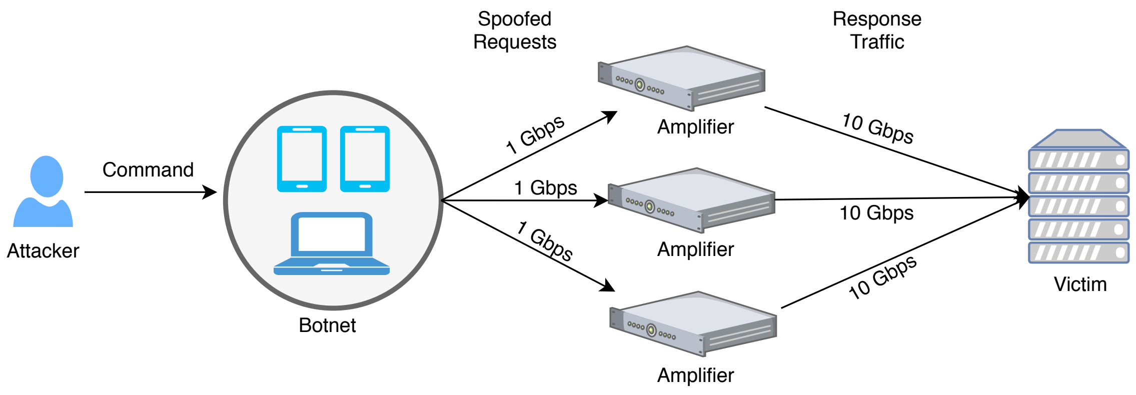


Figure 2.2: Bandwidth amplification attack using a botnet.

The bandwidth amplification attack is quite attractive to attackers due to several reasons. Firstly, it hides the attacker's identity as the victim receives backscatter traffic from the amplifiers, this makes it improbable for the victim to identify the attacker. Secondly, the attacker can easily launch a highly distributed attack because there are numerous publicly available network services that can be used as amplifiers. Using a small amount of available bandwidth, an attacker can generate traffic of several folds to exhaust the victim's bandwidth. Lastly, the amplifiers are UDP based and they don't have any source authentication mechanism which makes it possible for the attacker to reflect the traffic towards victim by spoofing source IP address.

2.3.1 Bandwidth Amplification Factor

Bandwidth Amplification Factor (BAF) is used as a measure for the potential effect of a bandwidth amplification attack. It is a ratio of the size of the response payload and the request payload. In this report, we use only the size of UDP request and response payloads in BAF calculations. The lower layer headers are not included so that our calculations of BAF remains valid in case the size of lower layer header changes in future. BAF is formally defined as:

$$\text{BAF} = \frac{\text{len(UDP response payload)}}{\text{len(UDP request payload)}}$$

2.3.2 Role of UDP in Bandwidth Amplification Attacks

User Datagram Protocol (UDP) is a simple, connection-less, message-oriented transport layer protocol. It does not guarantee reliable delivery of messages and retains no state of the messages once they are sent. Due to these features, UDP is widely used by streaming applications, DNS, NTP etc. When two hosts communicate using UDP as a transport layer protocol, they don't perform a handshake before initiating the communication. This query-response model of UDP renders it vulnerable to carry out amplification attacks.

In case of a reliable, connection-oriented transport layer protocol such as TCP, the client has to perform a handshake with the server, which makes it impossible to reflect the traffic towards the victim. If a client spoofs the IP address in the SYN packet, the server will send an SYN-ACK packet towards the target, the target can then respond with an RST packet or not reply with any packet at all. Since the size of an SYN-ACK packet is quite small, a spoofed SYN flood could not exhaust the victim's bandwidth.

Therefore, in order to exhaust the victim's bandwidth, the attacker sends a spoofed request to a UDP based service to trigger a response to the victim. A large number of publicly available UDP based services respond with response sizes that are much larger than the request sizes. Attackers exploit this disparity in request and response sizes and the connection-less characteristic of UDP protocol to generate amplified network traffic towards the victim. In chapter 3 we describe some of the well-known UDP based protocols which are abused to launch bandwidth amplification attacks.

2.3.3 Role of IP Address Spoofing in Bandwidth Amplification Attacks

One of the main requirements for launching a bandwidth amplification attack is to spoof the source IP address. IP address spoofing is a technique in which an attacker replaces the source IP address with a fake one for the purpose of disguising the identity. The Figure 2.3 shows an example scenerio for IP address spoofing. In case of bandwidth amplification attacks, the attacker spoofs the target's IP address and sends the request packets to UDP based services. The lack of source identity verification in UDP leads to immediately sending a response to the target. Using the IP address spoofing technique, the attacker becomes successful in reflecting traffic from the amplifiers to the victim. A large number of unsolicited responses entering a victim's network could exhaust its bandwidth and render the victim's services useless for legitimate users.

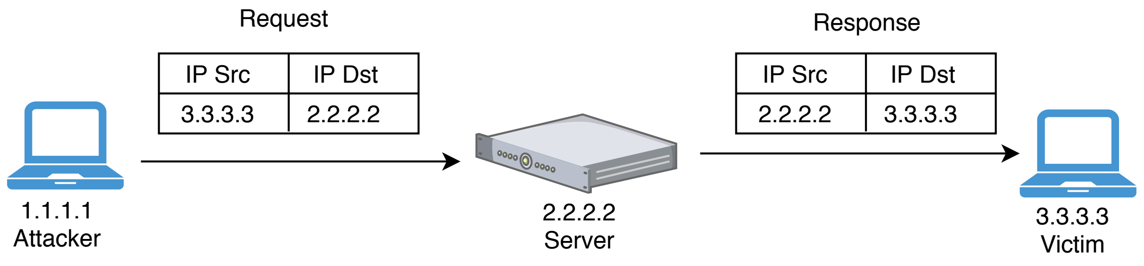


Figure 2.3: IP address spoofing example scenerio.

IP address spoofing is considered as a side effect of the design of the internet. Due to the dynamic nature of the packet routing rules, multihoming between networks and the packet routing asymmetry, it is impossible to verify if the packet is originating from a legitimate source. BCP38 [15] mentions ingress filtering rules which could be helpful in prohibiting an attacker to inject packets with forged source IP addresses in the network. Packet filtering rules can help to prevent the IP address spoofing problem, only if they are applied closer to the source i.e. implemented by the ISPs. The Spoofer project [40] conducts tests for detecting if a network allows sending packets with spoofed source addresses. The recent results published by them shows that around 20% of the total autonomous systems are spoofable.

Chapter 3

Bandwidth Amplification Attacks

In this chapter, we describe some of the UDP based protocols that are abused for amplification. We briefly describe the protocols and the attack vectors. We used our framework to perform scans for detecting the amplifiers mentioned in this chapter.

3.1 DNS Amplification Attack

DNS is a hierarchical and decentralized system which consists of thousands of servers all across the globe. DNS protocol uses port 53. The main use-case of DNS is to manage the namespace of the internet. Computer systems are identified by IP addresses, it is difficult for humans to memorize the IP addresses in order to access different sites. Moreover, the IP addresses are dynamic and they change over time. DNS solves this problem by helping the users to resolve domain names to IP addresses. Before going into the details of DNS based amplification attacks, it is important to explain the two major components of DNS; Name Server and Resolver.

As defined in the RFC 1034 [28], name servers are server programs which hold a set of information of the domain tree. Name servers are responsible for providing complete information for only a subset of the domain space, in case if they don't have the information, they provide pointers to other name servers that can lead to the final name server hosting the information. Resolvers are programs which fetch information from name servers in response to a client's request. In order to resolve a domain name to IP address, resolvers often contact multiple name servers and cache the results for performance reasons. The two variants of the DNS amplification attack are described below.

3.1.1 Abusing the Resolver

DNS provides support for TCP to deliver responses of large sizes. Due to TCP's connection establishment overhead and DNS's simple query-response model, UDP is often considered as a better transport layer protocol for DNS. To address the problem of depleting space for flags and additional data in DNS packets, an extension mechanism EDNS0 [46] was introduced to support UDP responses up to 4096 bytes. This gives rise to another problem, the attackers are now able to abuse DNS's *ANY* request with EDNS0 to generate larger responses for a request of small size. The DNS's *ANY* query is used to get all records available for a domain name.

The attackers aim to maximize the response size from DNS resolvers, for this purpose the attackers search for domain names which have many DNS records associated with them. Attackers could also configure their own domains and associate several Resource Records (RR) in the authoritative nameserver of their domain. Attackers often exploit the TXT record to inflate the DNS response size. The attackers spoof the source address in their requests to DNS resolvers to reflect large responses to the victim. Attackers usually use several publicly available DNS resolvers and command a botnet herd in order to launch a bandwidth amplification attack.

3.1.2 Abusing the Name Server

DNS spoofing also knows as DNS cache poisoning is a type of attack in which an attacker tries to introduce invalid or malicious mappings between domain names and IP addresses in a DNS resolver's cache. When the clients send requests to the DNS resolver, they receive the attacker's injected IP addresses. This results in redirecting the traffic to the attacker's site, enabling the attacker to eavesdrop and perform a man-in-the-middle attack. To defend against the DNS cache poisoning attacks, DNSSEC [1] was introduced to help the clients to validate the authenticity of a DNS response.

In DNSSEC deployments, additional RRs carrying digital signatures are provided to the client, this helps them to verify the authenticity but also increases the response size. DNSSEC addresses the problem of forged DNS responses but at the same time, it renders DNS name servers as attractive amplifiers for the attackers. It must be noted that the attackers send the DNS *ANY* request only to the authoritative name server because the non-authoritative servers respond with a small response size containing pointers to other name servers that may provide the information. Therefore, in order to scale the attack, the attacker has to send spoofed requests for domain names to their respective authoritative name servers.

3.2 NTP Amplification Attack

Network Time Protocol (NTP) is used for clock synchronization between computer systems which are connected over a network. NTP server listens on port 123. NTP follows a client-server model, clients contact the server to synchronize their clocks. Similar to the design of DNS, NTP also follows a hierarchical structure [26]. Each layer of the structure is called a stratum and it consists of several NTP servers.

The servers in a stratum can use their peers or server in the above stratum to synchronize their clocks. Clock synchronization is important for sequencing operations and events on distributed systems, without clock synchronization the computer systems would experience a clock drift which would cause the failure of services provided by the distributed system. Hence, NTP plays a critical role in the architecture of the internet. NTP is a UDP based network protocol, it is often abused for bandwidth amplification attacks.

NTP servers provide a *monlist* command to get a list of hosts that have recently queried the NTP server for clock synchronization. This command is usually used by the system administrators for debugging purposes. The big response from the *monlist* command could be exploited by the attackers for bandwidth amplification attacks. The number of hosts provided by the *monlist* command is capped at 600. The *monlist* command is enabled by default in all NTP version prior to version 4.2.7. It should be explicitly disabled by the admins or the NTP server should be upgraded to the latest version. The number of publicly available NTP servers on the internet and the large response size provided by the *monlist* command renders NTP servers as attractive amplifiers for the attackers.

3.3 QOTD Attack

Quote-of-the-day (QOTD) is a legacy protocol which is used for testing and debugging purposes. The protocol is defined in RFC865 [36], it is used to send a random quote to the client. QOTD uses port 17 and provides a TCP and a UDP interface. The support for responding to UDP datagrams makes it vulnerable to amplification attacks. The RFC865 specifies that the length of the quote should be less than 512 characters. Since the protocol simply discards the contents of the request payload, it can theoretically provide an amplification factor of more than 500. Response from one of the publicily available QOTD server is shown in the Figure A.3.2.

3.4 Chargen Attack

Chargen stands for character generator, the protocol is defined in RFC864 [6]. The protocol is similar to QOTD in a sense that it sends data without regard to the input. Chargen exposes UDP and TCP interfaces on port 19. Every time the UDP interface receives a request, it responds with a datagram of length between 0 and 512, containing random characters in the payload. Similar to QOTD, Chargen is also used for testing and debugging purposes. Response from a Chargen sever containing random ASCII characters is shown in the Figure A.3.3.

3.5 Memcached Attack

Memcached [25] is a caching system which is used to reduce the load on the database of web applications. It uses port 11211. Memcached is opensource and it can be deployed on Unix or Windows-based systems. Memcached maintains a distributed hashtable in memory, which spans multiple machines. It has a client-server architecture where the Memcached server provides a simple API to insert and retrieve records in the hashtable. Memcached provides both, UDP and TCP based communication interfaces. Prior to version 1.5.6, Memcached has the UDP protocol enabled by default. Publicly available Memcached deployments were therefore exploited by attackers to launch amplification attacks [39].

A publicly available Memcached server with the UDP protocol enabled could be exploited in several ways. In the most basic way, an attacker could send commands such as *stat items* to print statistical details about all of the items stored in the hash table. The response of such a command is quite large compared to the request size. Hence, attackers spoof the requests to trigger larger responses from Memcached server to launch a bandwidth amplification attack.

A more complex and devastating version of Memcached based amplification attack is when an attacker implants data in the hash table. The maximum size of the value allowed by Memcached is 1MB. The attacker could insert a key with a 1MB data, spoofed requests to get the key can result in very high amplification factor. Github has reported that Memcached servers can generate amplification factors as high as 51,000 [39].

3.6 SSDP Attack

SSDP stands for Simple Service Discovery Protocol, it is a protocol used for discovery and advertisement of network services. It uses port 1900. SSDP protocol is used by devices such as printers, wifi access points etc. to allow them to be easily discovered by other hosts in the network. SSDP uses UDP as its transport layer protocol. When a new device joins the network, it finds out about Universal Plug-and-Play (UPnP) devices by sending messages to a multicast address on port 1900. Once the UPnP devices receive the search message, they respond with a UDP unicast message to provide detailed information about the services offered by them.

In addition to querying devices using the multicast addresses, a device could also be queried using a unicast request. The response data sent by a UPnP device is quite large as compared to the request. It contains detailed information about the services offered by the device. The devices don't check if the querying device belongs to the local network or not. A poorly configured firewall could expose UPnP devices to the internet and the attacker could exploit such publicly available devices for bandwidth amplification attacks.

3.7 RIP Attack

RIP stands for Routing Information Protocol, it is a legacy routing protocol which uses distance-vector routing technique and uses hop count as a metric. It uses UDP as its transport layer protocol and listens on port 520. The RIPv1 protocol lacks authentication mechanism, moreover, the protocol has been declared as deprecated in RFC1923 [2] because it only supports classful addressing. The low convergence time and the hop limit of 15 makes it less attractive to use in networks.

RIPv1 enabled routers broadcast their routing table by default after every 30 seconds, they also provide their routing table if they receive a request from their neighboring router. Due to the lack of authentication and the use of UDP protocol, the attackers send spoofed requests to RIPv1 routers, which respond with their routing table to the victim. The size of the response of RIPv1 routers is quite large, due to the disparity in request and response sizes, the attackers are able to launch a bandwidth amplification attack on the victim.

3.8 CLDAP Attack

CLDAP stands for Connection-less Lightweight Directory Access Protocol. It operates on port 389. Unlike the Lightweight Directory Access Protocol (LDAP) CLDAP uses UDP as its transport layer protocol. It is built on LDAP protocol and provides a restricted set of operations as compared to LDAP. The main motivation behind its development was to provide a way for applications to retrieve information from the directory service without the overhead of connection establishment.

An incorrectly configured and publicly available active directory service could be exploited by attackers. The attackers use the CLDAP protocol to send requests which trigger large responses e.g. *searchrequest* to get a list of all users registered in the directory service. The request packets are spoofed to direct the responses towards the victim. In this way, an attacker could exploit the CLDAP protocol to launch a bandwidth amplification attack.

3.9 NetBIOS Attack

NetBIOS stands for Network Basic Input/Output System, it is used for communication between applications running on a local area network. It operates on port 139. It provides both connection-less and connection-oriented interfaces for communication. The NetBIOS provides a name service to help in locating the resources identified by NetBIOS names. It also provides session service for reliable message exchange between applications and a datagram service for unreliable, non-sequenced, connectionless communication between applications.

NetBIOS name service should be strictly configured to allow name resolution requests from hosts in the local network. However, due to misconfigurations, a large number of NetBIOS devices are publicly available which could be exploited by the attackers. Similar to DNS, the NetBIOS name service does not support any authentication mechanism. An *nbtstat* query could be used to get a list of NetBIOS names of all NetBIOS applications running on a system. Such a request could be spoofed to generate traffic towards the victim to consume his bandwidth. Therefore, publicly available NetBIOS devices are frequently abused by attackers. A typical response to an *nbtstat* query is shown in the Figure A.3.1.

3.10 SNMP Attack

SNMP stands for Simple Network Management Protocol and it uses port 161. SNMP can be used to fetch the system status, configurations and other variables remotely from a devices such as printers, routers, video cameras etc. SNMP can also be used to set configurations on these devices. The SNMP managed network consists of agents and managers. Agents runs on the managed devices and listen for SNMP requests from the managers. The Managers are used to set or get information from the agents.

SNMP enforces authentication with the use of community strings. A community string could be thought of as a passphrase to access the statistics of a device. SNMP version 1 and 2 support community strings whereas version 3 supports username and password based authentication. SNMPv1-v2c ships with a default community string *public* and the information about the device is provided in a read-only mode by default. It is assumed that administrators would configure their devices and change the default community strings.

Attackers exploit the publicly available SNMP devices which have a *public* community string. The attackers try to find an SNMP *GetRequest* for a system property which maximizes the response size. Generally, the attackers fetch the system description property to generate a large response as shown in the Figure A.3.4. A much worse amplification attack happens when the attacker uses a *GetBulk* request. The *GetBulk* operation was added in SNMPv2 and it is used to get a list of properties with one request packet. The large response sizes and the default passphrases renders the SNMP devices abusable for amplification attacks.

Chapter 4

Framework

In this chapter, we explain the design and architecture of the framework. The framework is designed for executing internet-scale network measurements for detecting potential amplifiers. The framework leverages active network measurements for this purpose. We discuss the major features provided by the framework, the general architecture and the technologies used to build the backend and the frontend. Moreover, we explain how the selected tools and technologies have helped to increase the scalability and the usability of the framework.

The network measurements are performed for several purposes ranging from the performance evaluation of networks to detecting malicious hosts and traffic. To achieve these goals, the network administrators have the options of either actively or passively scanning the networks. In case of active scanning, probe packets are sent to the hosts in the network, the responses to the probes are then analyzed for gathering useful insights. Active scanning can be used for gathering information about the network topology, performance e.g. throughput, and latency and to perform internet-wide measurements.

In passive network scanning, the network traffic flow is observed using monitoring probes in the network. Passive network scanning can be used to measure traffic volume, network resource utilization, intrusion detection etc. Network administrators choose the type of scanning technique based on the information they want to gather, in some cases, a hybrid approach is used which involves both active and passive network measurements.

Our goal is to detect potential amplifiers before an amplification attack takes place. Passive measurements can help to identify the amplifiers using key indicators such as the packet size similarity, similar payloads in outgoing and incoming network traffic and a large number of ICMP error messages [3]. However, it should be noted that the passive measurements can only detect the amplifiers once an attack is launched and the traffic is analyzed. Our framework makes use of active network measurements and is capable of detecting and alerting for potential amplifiers before the attack takes place.

4.1 Design Goals

The basic idea behind our amplifier detection technique is to probe the network with different request payloads that trigger a response packet which is larger in size than the request packet. We have one scan for every type of amplifier we are interested in detecting. For example, for detecting NTP amplifiers we have a scan which sends NTP *monlist* request packets in the network. Similarly, to detect DNS based amplifiers a different scan is used to send DNS *ANY* type queries for a given domain. To maintain an up-to-date list of active amplifiers, the framework is capable of scheduling scanning jobs and executing them at the specified intervals. The results of the scans are then processed and stored in a database and they can also be visualized using a dashboard. The five major design goals addressed by the framework are discussed in following subsections.

4.1.1 REST API and CLI Client

The framework provides support for a REST API to communicate with the backend in an efficient way. A number of API endpoints are exposed to facilitate the user to create and manage the network scans. Moreover, the results of the network scans can also be fetched from the backend using the REST API. The REST API is developed using the Django REST Framework [11].

To increase the usability of the framework a CLI client is also developed for interaction with the framework. The CLI client validates the user's input and responds with meaningful messages to assist the user in his interaction with the framework.

4.1.2 Scheduling Scanning Tasks

In order to keep track of active amplifiers, it is important to run the scans at regular intervals. For this purpose, the Celery Beat scheduler [9] is used in the framework to provide support for scheduling scanning tasks. The Cron format is used to specify the time and frequency of a scanning task.

Cron is a popular format among system administrators, it not only increases the usability of the framework but also provides a powerful and flexible way to specify the scheduling information. In regards to the ethical considerations of performing network scans, the framework also supports the features of disabling/enabling a specific scanning job and purging a running scanning job.

4.1.3 Fast Execution of Network Scans

Network administrators often use Nmap scanner [32] for host discovery, port scanning and finding vulnerabilities in the network. Nmap is considered as a mature and state of the art tool. It keeps state for every packet in transit which makes it unsuitable for executing internet-wide scans. One of the use-cases of our framework is that it should be able to detect amplifiers on the internet-wide scale.

Therefore, the framework makes use of ZMap which is a fast network scanner used for internet-wide measurements. It is claimed and verified that using 1 Gbps of network bandwidth, ZMap can scan the entire IPv4 address space in under 45 minutes [12]. The framework is also capable of scanning the IPv6 address ranges, for this purpose we used the IPv6 UDP probe for ZMap [22].

4.1.4 Visualization Dashboards

The framework provides several dashboards to visualize the scanning results. Grafana [20] is used as a tool for data visualization. The Grafana server directly communicates with the database to access scan results. Time-series visualizations are generated to analyze the change in the number of active amplifiers and the change in BAF of an amplifier over a period of time. The visualization dashboards can be used identify the worst amplifiers. More details about the visualizations are shared in the section 4.2.7.

4.1.5 Notification Module

The framework provides support for sending e-mail notifications. It is important to notify the administrators about the detected amplifiers so that appropriate actions can be taken to mitigate the amplification threat. For this purpose, the e-mail notifications are sent to the administrators which describe a list of detected amplifiers and their bandwidth amplification factors.

In case a scanning job fails due to some exception, the complete details about the failure are also communicated via e-mail notifications. The failure details help the administrators to fix the issue before the next scanning iteration takes place. The administrators could also temporarily disable the scanning job while they fix the issue. Hence, the notification module not only increases the usability but also plays an important role in tackling the operational challenges of the framework.

4.2 Design & Architecture

In this section, we explain the design and architecture of the framework. For the development of the framework, we used Python as a programming language. Python is an interpreted high-level programming language. It is widely used these days in scripting and automating computing tasks to developing complex software systems. It supports multiple programming paradigms such as object-oriented, imperative, and functional. It has a comprehensive standard library and has a very active community of developers. Python has been ranked as number one programming language in 2018 by IEEE [5].

Our main motivation for using Python was the countless libraries, apps, and plugins available which could speed up the development process. Instead of reinventing the wheel, our goal throughout the development phase was to re-use the existing technologies. Most of the technologies that we used have performed well on production scale deployments. The use of Python libraries have rapidly decreased the development time without compromising on the software quality.

4.2.1 REST API

REST stands for **RE**presentational **S**tate **T**ransfer, it is an architecture style that defines a number of constraints to be used for creating web services. It was authored by Roy Fielding in 2000 in his Ph.D. dissertation [16]. Web services that follow the RESTful architecture style provide interoperability between computer systems on the Internet. To create a RESTful web service, it must follow the following six constraints.

1. Client-server architecture:
   Client-Server architecture enforces the idea of *"separation of concerns"*. The decoupling of the client and server allows them to evolve independently. To access the resources on the server the client should only know about the resource URIs. This improves the portability of the user interface across multiple platforms and also improves scalability by minimizing the coupling between a server and a client. The Figure 4.1 shows a client-server communicating using the REST architectural pattern.

2. Statelessness:
   Statelessness constraint specifies that requests from the client should contain all the information necessary to serve the request. No client context is stored on the server, if a stateful behavior is required then the state must be maintained in client and passed down to the server along with the request.

3. Cacheablility:
   The cacheability constraint specifies that the response from the server should be implicitly or explicitly labeled as cacheable or non-cacheable. If the client caches the responses, it would reduce the load on the server and increase the performance significantly.

4. Layered system:
   RESTful web service allows using a layered architecture where the API is deployed on one server and the data is stored on another. The client can't distinguish if it is directly connected to an end server or an intermediate server. The client is agnostic to the number of layers between itself and the actual server that is responding to the request.

5. Uniform interface:
   The fundamental constraint of the REST architectural style is the uniform interface. It specifies that the resources should be identified using logical URIs, the resources should not be large in size and they should contain information to modify the resource or fetch the additional information. Moreover, the resources should be served in a consistent representation across the system, they should follow a strict data format e.g. JSON or XML.

6. The code on demand (Optional):
   The code on demand is an optional constraint, it describes that in addition to resources in XML and JSON formats, the server could also provide executable code to the client to extend its functionality.

API Endpoint	**Request Method**	**Description**
/api/v1/scan	POST	create new scan
/api/v1/scan	GET	list scan names
/api/v1/scan/{name}	GET	get scan details
/api/v1/scan/{name}	PUT	update scan
/api/v1/scan/{name}	DELETE	remove scan
/api/v1/scan/{name}/result	GET	get latest scan result
/api/v1/scan/running	GET	list task-ids of running scanning jobs
/api/v1/scan/revoke/{task_id}	GET	revoke running job with the specified task_id

Table 4.1: REST API endpoints

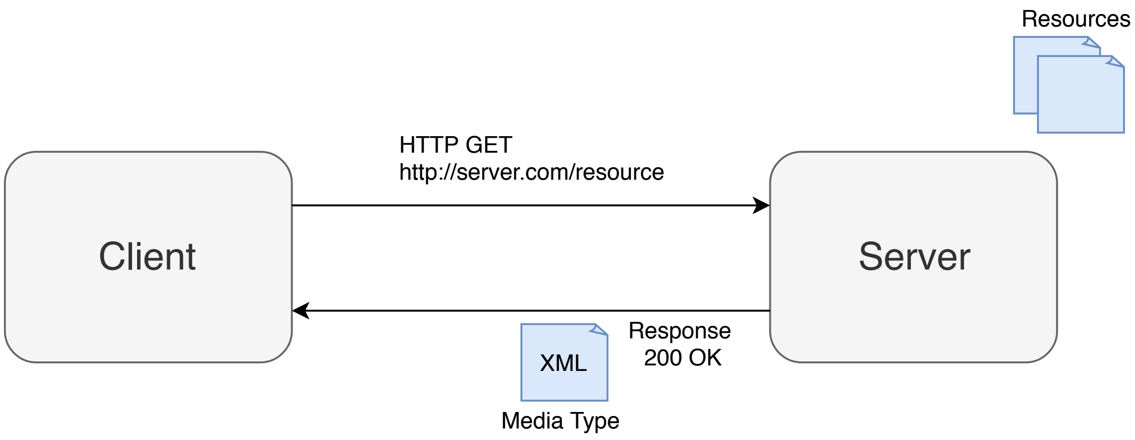


Figure 4.1: Client-Server interaction in REST architecture style

4.2.2 Django REST Framework

For the development of REST API, we used the Django REST Framework (DRF). Before diving into the details of DRF, we will first introduce the Django web framework. Django [10] is a Python based, opensource web frameworks, which follows a Model-View-Controller (MVC) architecture pattern [24]. The Django framework focuses on rapid development, pluggability of components, less code and less code coupling principles. Django provides support for a database abstraction layer, built-in form validations, and support for a lightweight development server. DRF is built on top of the Django web framework to support the development of RESTful web services.

We used DRF to develop the API endpoints mentioned in Table 4.1. The request parameters are checked against several validators e.g the passed address range should conform to the format of an IPv4 or IPv6 address and the target port number should be between 1-65535. In case if an API request does not pass through the validation checks in the backend, meaningful error messages are returned to the user for assisting them to correctly construct a request.

4.2.3 Command-line Client

To increase the usability of the framework, we have also developed a command-line client in Python. CLI client provides support for several commands for creating, updating, deleting or fetching results of a scan. As shown in the Figure 4.2, the CLI client receives the commands from the user, checks if all the required arguments are supplied by the user and then sends the request to the framework. The CLI client helps to avoid sending requests with incomplete arguments to the backend. In the case of a successful response from the backend, CLI client parses and prints the information in a human-readable format instead of displaying the JSON data. CLI client also formats and displays the errors returned by the server. The appendix A.1 shows the examples of CLI commands and their outputs.

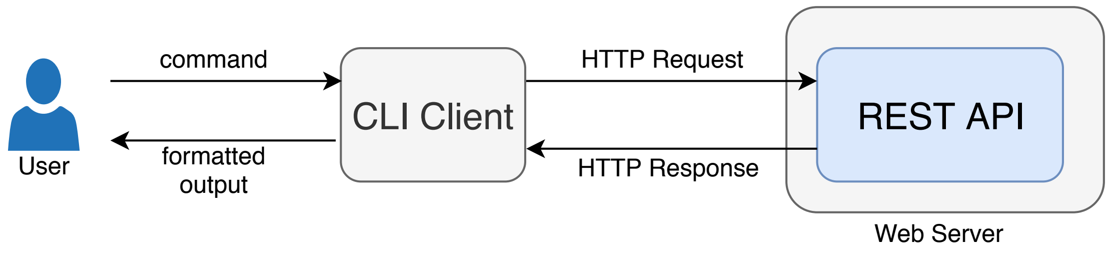


Figure 4.2: CLI client's interaction with the framework backend

4.2.4 ZMap Network Scanner

One important component of the framework is the network scanner. Generally, a network scanner is used to gather information about the services offered by different hosts in a network. Our motivation for using the network scanner is to detect amplifiers in a network by sending specific request packets which could trigger large responses. We have chosen ZMap to execute the network scans. In this section we have explained our intuition behind selecting ZMap as the network scanner.

Traditional tools such as Nmap are widely used for vertical scans i.e. scanning of multiple ports on a single host. Nmap supports a variety of features including operating system detection, port scanning, scripts for vulnerability detection etc. However, it is not a suitable network scanner for our framework because of its stateful design. Nmap is slow for large scale scans because it keeps track of the packets in transits and waits for the responses to arrive. Due to the low performance of executing a large scale scan, Nmap was not a suitable option for our framework.

As our goal was to develop a framework that is capable of detecting amplifiers on a large scale, we chose the ZMap scanner which is capable of scanning complete IPv4 address range in under 45 minutes [12]. ZMap is suitable for executing horizontal scans i.e. scanning of multiple hosts for the same port. Horizontal scans are used to gather information about the distribution of vulnerabilities and security audits on a global scale. ZMap's design focuses on executing a fast scan on a large scale. For this purpose, ZMap employs a stateless design and it doesn't store any per-connection state. ZMap maintains separate threads for sending requests and receiving responses, this greatly improves the overall performance because the sending thread does not block for packet responses.

To further increase the performance, ZMap completely bypasses the TCP/IP stack and generates to Ethernet frames directly. ZMap uses techniques similar to SYN Cookies to determine if the response packet is intended for the ZMap or not. To avoid saturating the networks, unlike Nmap which adapts the transmission rate, ZMap relies on randomly selecting targets to send probes. ZMap by default sends one probe per target, it does not detect connection timeouts to retransmit packets. Experiments have shown that ZMap achieves 98% network coverage [12] using only a single probe per host. In addition to scanning the IPv4 address ranges using ZMap, the framework also makes use of a probe module [22] developed for ZMap by TUM's chair of Network Architectures and Services which provides support for scanning IPv6 address ranges. Due to these features, we considered using ZMap as a network scanner for our framework.

The REST API is provided by the framework to create scans to detect different types of amplifiers. While creating the scans, the target address ranges, port number, request payload and the packet rate per second are provided as request arguments. ZMap's UDP probe module is then used to execute the scan using the supplied arguments. As ZMap focuses on horizontal scans, it does not provide support for executing a scan targetting multiple port numbers, therefore we run one ZMap scanning job per type of amplifier we are interested in detecting.

ZMap provides several output modules such as csv, JSON, Redis etc. to output scan results, we use the ZMap's csv output module in our scanning jobs. The csv output is later parsed and persisted in a database. ZMap supports both whitelisting and blacklisting of address ranges. We use the ZMap's blacklist to avoid scanning the local, reserved and multicast addresses. Moreover, in case of a complaint from network administrators, we could simply add their network addresses in ZMap's blacklist to avoid scanning them. This also helps in maintaining the ethical considerations of scanning the networks.

4.2.5 Scan Scheduling

One important component of the framework is the scan scheduler. It is required that the scans are executed periodically to detect the amplifiers. When the users create scans using the REST API, they provide arguments in cron format to express the intervals after which the scans should be executed. The framework provides support for executing the scanning jobs asynchronously. For this purpose, we used Celery [9] which is a task queue developed in Python. In this following subsections, we explain how we have integrated Celery with our framework.

Celery Task Queue

Celery is a simple, flexible and reliable distributed system which provides support of task queues for applications. Most of the times, applications have to execute operations which take longer to process such as sending emails, image resizing etc. Therefore, the user's request is returned immediately without blocking and the task is executed asynchronously. The user can later issue commands to fetch the results of the tasks. Task queues are quite useful to scale a system because they distribute tasks to worker processes running on multiple machines.

A task is a unit of work for a task queue, in our case running a scanning job using ZMap is a task. To deliver the tasks to the Celery worker processes, a messaging broker is used. A messaging broker technology is used to deliver messages between different parts of the same application or between different applications, it is used in celery to deliver tasks execution requests to the worker processes. A publisher-subscriber based model is used, the tasks to be executed are sent to the messaging broker, the Celery workers connect and consumer from the messaging broker. In this way, Celery workers are able to receive the tasks.

The use of Celery in the framework helps in scaling the system horizontally because new workers can be connected to the messaging broker to increase the framework's task execution capacity. It also increases the high-availability factor because if a worker shuts down, there are other workers available to the framework which can execute the tasks instead of bringing the system to a complete halt.

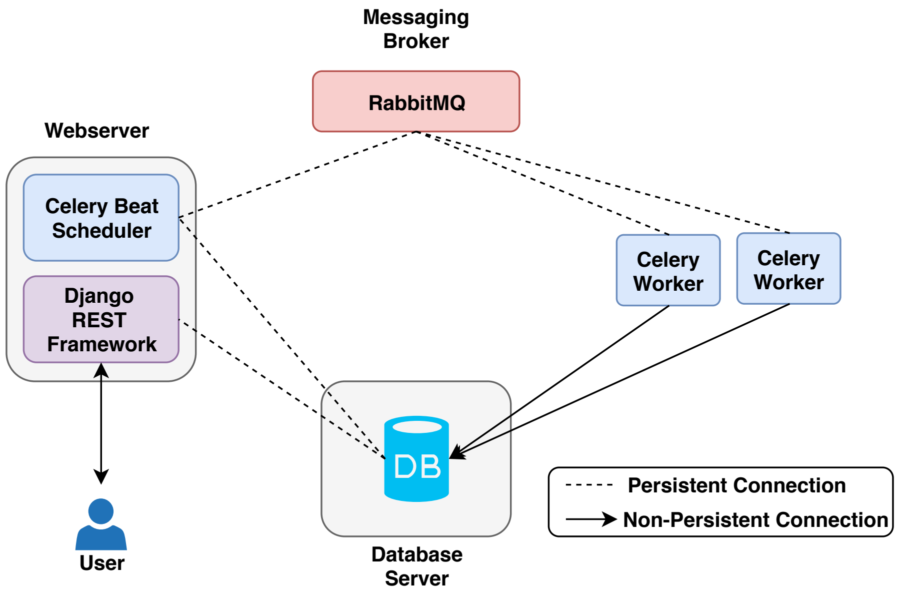


Figure 4.3: Architectural diagram of Celery's integration in framework.

Celery Beat Scheduler

Celery provides a scheduler to trigger the execution of tasks after the specified interval of time. The Figure 4.3 shows the high-level view of the Celery integration in our framework. As shown in the figure, the Celery Beat scheduler maintains a persistent connection with the database, it keeps track of the periodic task records i.e. scan entities created by users and triggers them at the specified interval. Celery Beat scheduler sends the task execution request along with the task arguments such as address ranges, port number, request payload etc. to a messaging broker. Redis [4] or RabbitMQ [45] can be used as a messaging broker. We used RabbitMQ because it is the default broker for Celery and it doesn't require any additional dependencies to make it work with Celery.

The Celery workers receive the task execution request along with the parameters for the scanning job from the messaging broker. The workers execute the scans using ZMap scanner. The results from the ZMap scans are parsed and stored in the database. The user can later use the REST API to view the details of detected amplifiers in a scan.

4.2.6 Processing Scan Results

The ZMap scan outputs the responses in a csv format. The Celery worker parses the responses and calculates the bandwidth amplification factor to filter out potential amplifiers. In order to store the results from tasks, Celery provides RPC result backend and a database backend. The RPC backend returns results to the messaging broker in form of messages that are non-persistent. In case a broker restarts, the results would be lost, that's why we have used the database result backend to persist the results in a database. The Celery task transforms the responses from csv format to a python dictionary, it serializes the result dictionary and stores it along with the information about task execution in a Task Result record.

After the creation of Task Result record, the backend creates a time-series record about the scan. The attributes of the time-series record can be seen in the Figure 4.4. Time-series records are important for visualizing the data using Grafana. The details about the amplifiers such as address, amplification factor, responses received from the amplifiers etc. are stored in a database for future analysis. The Figure 4.4 shows the attributes that are persisted in the database.

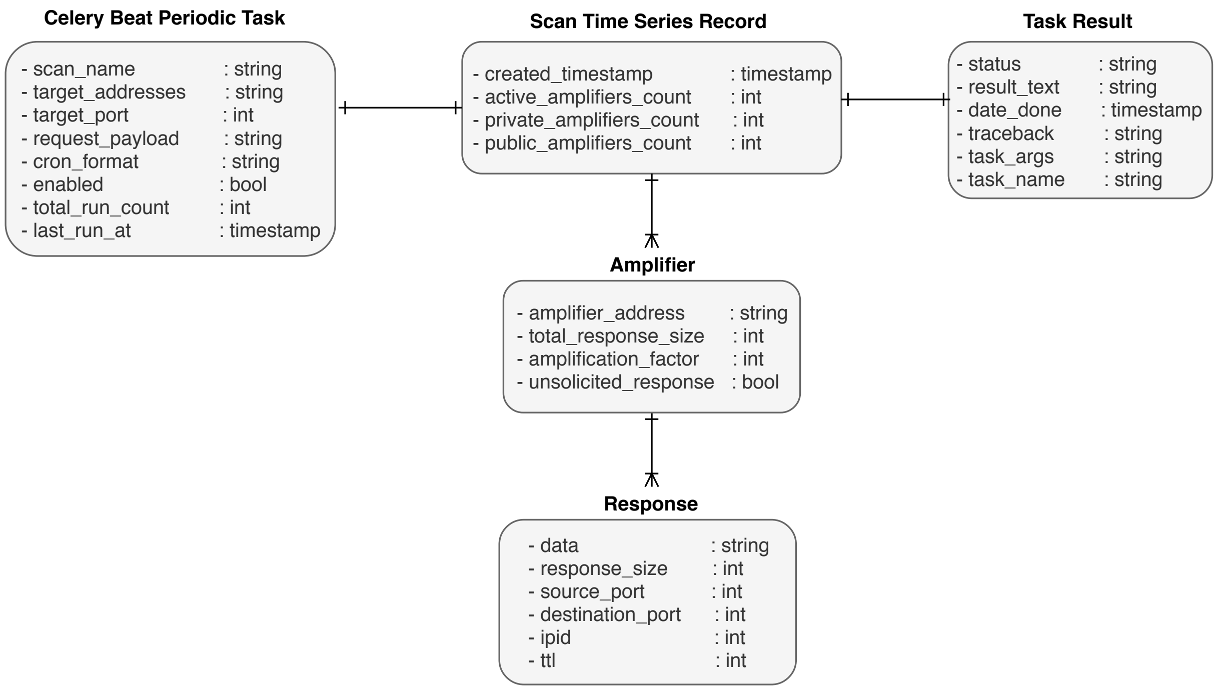


Figure 4.4: Entity relationship diagram.

4.2.7 Visualization

The framework provides support for visualizing results of scans. For this purpose, we have used Grafana to create several dashboards for gaining insights from the scanning results. Grafana is a completely open source technology, it provides the support of querying data from databases of several types. Moreover, the data from different data sources could be unified in a single dashboard. The widgets provided by Grafana such as heat maps, graphs, tables etc. make it easy to understand the data.

The Figure 4.5 shows how Grafana is integrated with our database. The Grafana WebUI running on a client device sends requests to a Grafana server. The Grafana server directly runs queries on the database and returns the data to the WebUI. WebUI polls the Grafana server to refresh the visualizations with the latest scan results. The refreshing timer is adjustable as per the needs of the system.

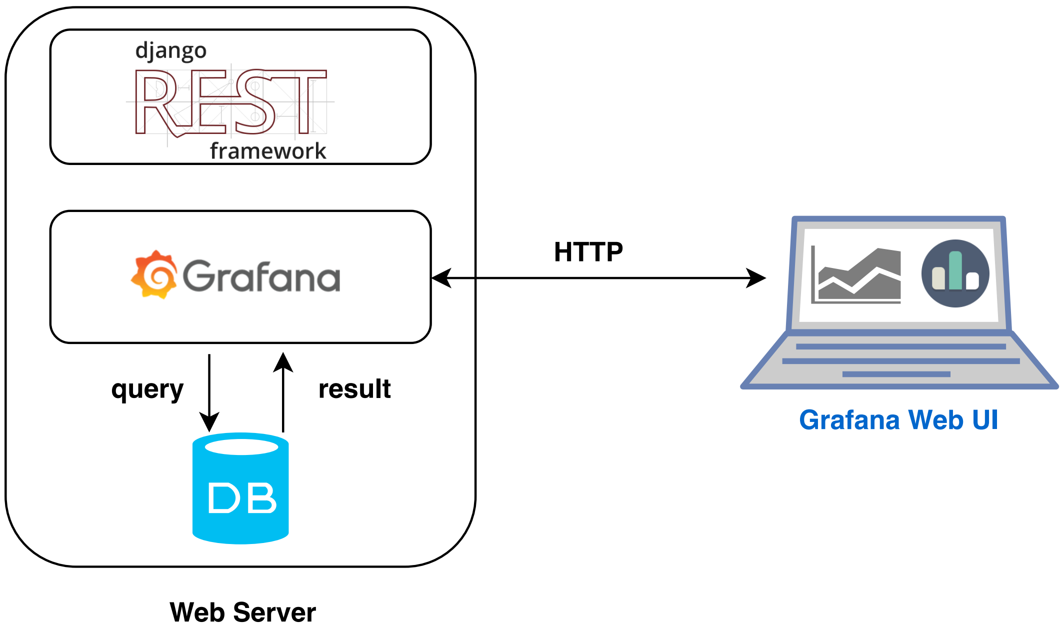


Figure 4.5: Grafana integration in framework.

We have used Grafana to develop three dashboards A.2. The *Home* dashboard shown in the Figure A.1 displays the total number of active amplifiers detected from all scans. It also displays the number of publicly and privately available amplifiers. The dashboard contains a graph for visualizing the change in the number of active amplifiers for all scans. Moreover, the table widget shows the addresses of active amplifiers, their bandwidth amplification factors and the type of scan of they belong to. The *Scan Details* dashboard shown in the Figure A.2 displays the total number of amplifiers detected in a scan of a specific type. It also displays the number of times the scan was performed and the change in the number of detected amplifiers is also displayed in a graph. Lastly, the *Amplifier* dashboard shown in the Figure A.3 displays the details about a specific amplifier. It presents the change in BAF of the amplifier and the information about response packets received from the amplifier.

4.3 Scan life-cycle

The steps involved in the creation and execution of a scan are described below .

1. Scan creation:
   The Python client is used to create a scan to detect a particular type of amplifier. The scan attributes include a list of address ranges to be scanned, the targeted port number, the request payload, the scan execution interval, and the packet rate per second. If the scan creation request is valid, a new scan record is stored in the database.

2. Celery beat scheduler gets informed:
   When a new scan is added to the database, the scheduler updates its internal memory and starts tracking the last runtime of the scan. When the time arrives for the scan execution, the celery beat scheduler sends the task execution request to the messaging broker.

3. Celery worker executes the scan:
   The Celery worker consuming from the messaging broker receives the request for task execution. The worker uses ZMap to execute the scan using the provided arguments.

4. Processing scan results:
   Responses from ZMap scan are processed to find hosts with BAF greater than one. The responses received from such hosts are stored in the database.

5. Email notification:
   The Celery worker sends an e-mail to the administrators to notify them about the detected amplifiers and their bandwidth amplification factors. In case, if the scanning job fails, an e-mail is sent to describe the encountered error.

6. Grafana visualizations:
   At this point, the database contains entries from the latest scan run. Therefore the grafana WebUI refreshes the dashboard with the latest data.

7. Scan termination:
   A scan could be deleted by the user, in that case, all the results from the scan's iterations are also removed from the database. A scan could also be temporarily terminated by using the scan *disable* command. If the scan's status remains enabled, the Celery Beat scheduler would trigger the scan execution at the next interval.

Chapter 5

Measurements

In this chapter, we discuss the results of active measurements for the detection of amplifiers. We use our framework to schedule scans for detecting protocols which are abused for amplification. A brief summary of these protocols is described in chapter 3. The next sections explain the details about the measurement setup, scan configurations and the analysis of the scanning results.

5.1 Munich Scientific Network

Munich Scientific Network (MWN) [35] connects together Bavarian state library, museums, student residence halls, various educational institutes such as the Hochschule München (Munich University of Applied Sciences - HM), the Technische Universität München (TUM), the Ludwig-Maximilians-Universität München (LMU), the Bayerische Akademie der Wissenschaften (Bavarian Academy of Sciences and Humanities - BAdW), the Hochschule Weihenstephan-Triesdorf (Weihenstephan-Triesdorf University of Applied Sciences - HSWT), and research centers of the Max Planck and the Fraunhofer Society.

The MWN provides connectivity with the help of a backbone network of switches and routers. The Figure 5.1 shows the networking infrastructure of the MWN. Depending on the volume of communication and the location size, MWN provides bandwidths ranging from 100 Mbps to 100 Gbps. MWN employs fiber optic cables for offering a high-speed physical medium. These fiber optic cables are rented from Deutsche Telekom and M-Net. The Leibniz Supercomputing Centre (LRZ) is responsible for operating the MWN.

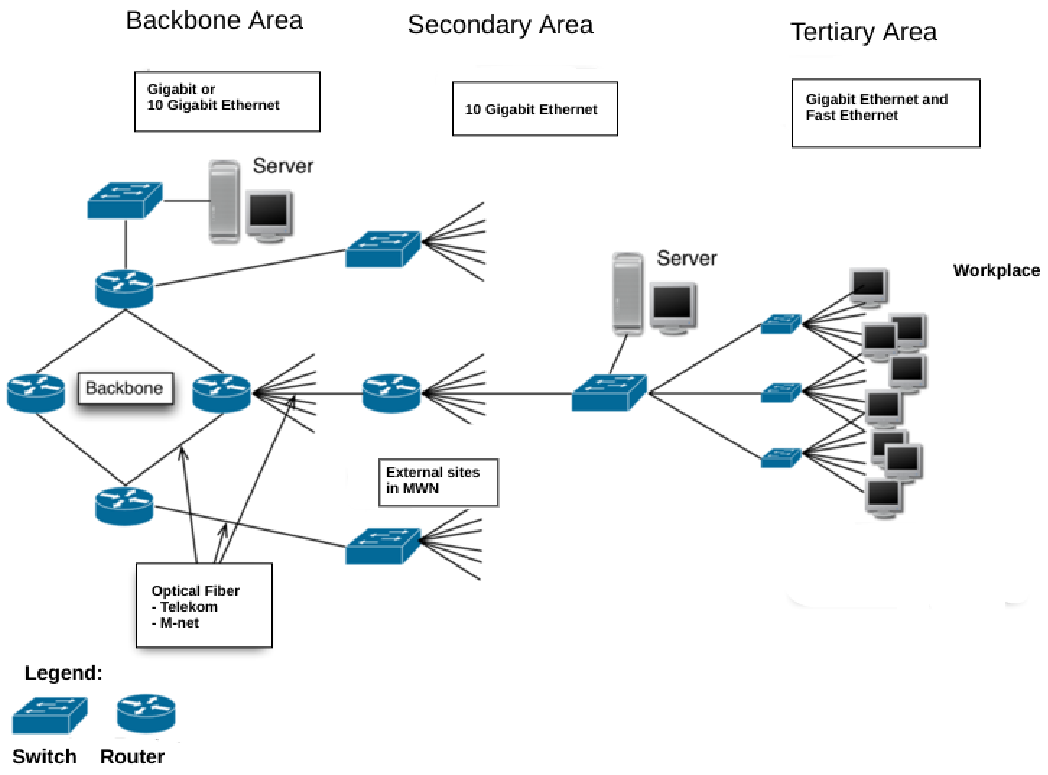


Figure 5.1: Networking infrastructure of Munich Scientific Network based on [29].

5.1.1 Scanned Address Ranges

In order to evaluate our framework in a real-world scenario, we scanned a subset of the address ranges of MWN for the detection of amplifiers. Our scans targeted the public IPv4 address ranges belonging to TUM. We scanned 317 subnets containing a total of 130,810 addresses. Moreover, we configured a blacklist to ignore the IP addresses belonging to IANA's reserved address ranges. Considering the privacy and security risks, explicit details about vulnerable hosts and subnets are not described in this report.

5.2 Measurement Setup

Our measurement setup shown in the Figure 5.2 consists of two machines, a master node, and a scanning node. The hardware specifications and the services running on the two machines are described below.

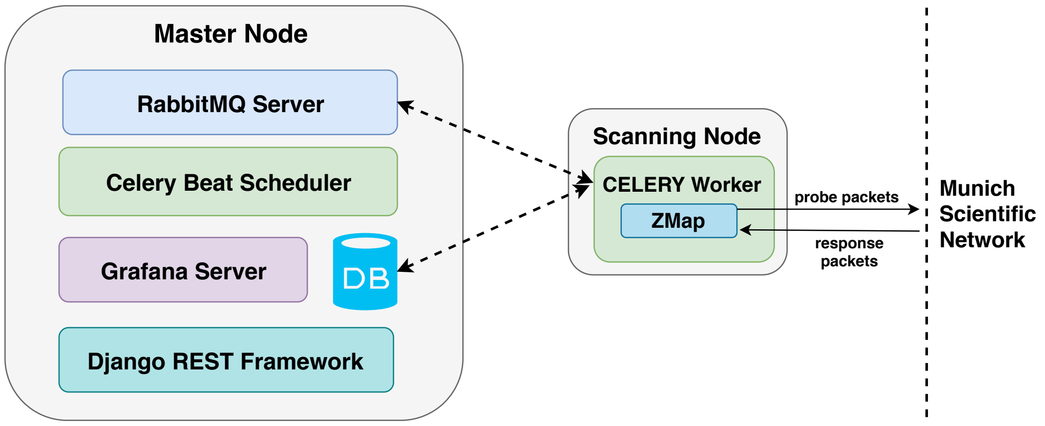


Figure 5.2: Measurement setup for scanning MWN.

5.2.1 Master Node

Master node is a virtual machine which is allocated a single core with 4 threads and 12 GB of RAM. The VM runs Debian GNU/Linux 9 (stretch) with kernel version 4.9. The master node acts as a web server and exposes the REST API endpoints to the user. It runs the MariaDB server 10.1.37 to provide support for a relational database. The master node is also responsible for running the Celery scheduler, the Grafana server, and the RabbitMQ server.

5.2.2 Scanning Node

We used a single scanning node in our measurement setup. The scanning node is equipped with a quad-core 3.20GHz Intel Xeon CPU with hyperthreading, 12 GB of RAM and Intel's 1 Gigabit Ethernet network card. The scanning node runs Celery worker daemon, which executes the network scans using ZMap. It must be noted that only one Celery worker process was spawned in the scanning node to enforce the execution of one scanning job at a time. Due to the ZMap's stateless design, we discourage the execution of parallel scans on a single machine. In order to scale the system by running scans in parallel, we recommend adding new scanning nodes with one celery worker per scanning node.

5.3 Scanning Configuration

The Table 5.1 describes the 10 UDP-based protocols that we have assessed for their abuse against bandwidth amplification attacks. The scans were performed on the TUM's public IPv4 address ranges. We used the measurement setup described in section 5.2 to schedule the scans. As the number of active hosts increases significantly during the working hours, therefore to find out the change in the number of active amplifiers during a 24 hours period, we ran scans twice a day with a 12-hour interval. We have also configured a blacklist to avoid scanning the IANA's reserved address ranges.

5.4 Ethical Considerations

The following measures were taken to conform to the ethical considerations of running network scans:

1. Validation of request payloads
   We validated the request payloads against the protocol standards. We made sure that our scanning probes don't cause any problems on the hosts.

2. Describe scanning intentions
   We used dedicated servers with rDNS names to perform network scans. A webpage [43] was hosted to inform the administrators of scanned networks that the intuition behind our networks scans is scientific and we don't intend to perform any malicious activities.

3. Maintaining a blacklist
   We maintained a blacklist to avoid scanning the network addresses for whom we have received complaints. Our webpage provided abuse contact details to file a scanning complaint with us.

4. Avoid saturating networks
   For reducing the risk of saturating the network with scanning probes, we executed our scans with a low packet rate of 128 pps. The ZMap's optimized probe module also helped to avoid saturating network paths by selecting network addresses randomly.

5. Restricted access to scan results
   We have not published the IP addresses of vulnerable hosts because chances are that attackers might abuse them. Moreover, the code for the framework is kept in a private repository with access to limited personnel.

Protocol	Attack Vector
Chargen	get random ASCII characters
CLDAP	get list of users registered in a directory
DNS	DNS ANY query for google.com
Memcached	*stat items* command
NetBIOS	get NetBIOS name table
NTP	*monlist* command
QOTD	get random quote
RIPv1	get routing table
SNMP	get system description field using community string public
SSDP	discover packet for UPnP devices

Table 5.1: Overview of attack vectors used for detection of amplifiers.

Protocol	Min	Max	Avg
NetBIOS	73	215	122
SNMP	23	27	25
SSDP	1	3	2
Chargen	2	2	2
DNS	1	1	1

Table 5.2: The min, max and avg. number of amplifiers detected per protocol over a period of two weeks.

5.5 Scanning Results & Analysis

In this section, we describe the results and analysis of our network measurements. Figure 5.3 shows the results of executing scans for detecting 10 different types of amplifiers. We have not detected any publicly available amplifier for CLDAP, RIPv1, NTP, Memcached, and QOTD protocols. The number of active amplifiers for some protocols such as NetBIOS change significantly as compared to other protocols. Similarly, the Figure 5.5 shows that the BAF for some protocols varies with time. In the following sub-sections, we provide an in-depth analysis about the detected amplifiers.

5.5.1 NetBIOS Amplifiers

The highest number of amplifiers in our measurements belong to the NetBIOS protocol. NetBIOS is enabled by default on most Windows operating systems. We used Nmap's OS Detection utility to verify the operating systems of NetBIOS amplifiers. We found out that in addition to the Windows OS, there were a majority of Linux/Unix based hosts running NetBIOS with the help of SAMBA [41], which is a standard Windows interoperability suite of programs for Linux and Unix.

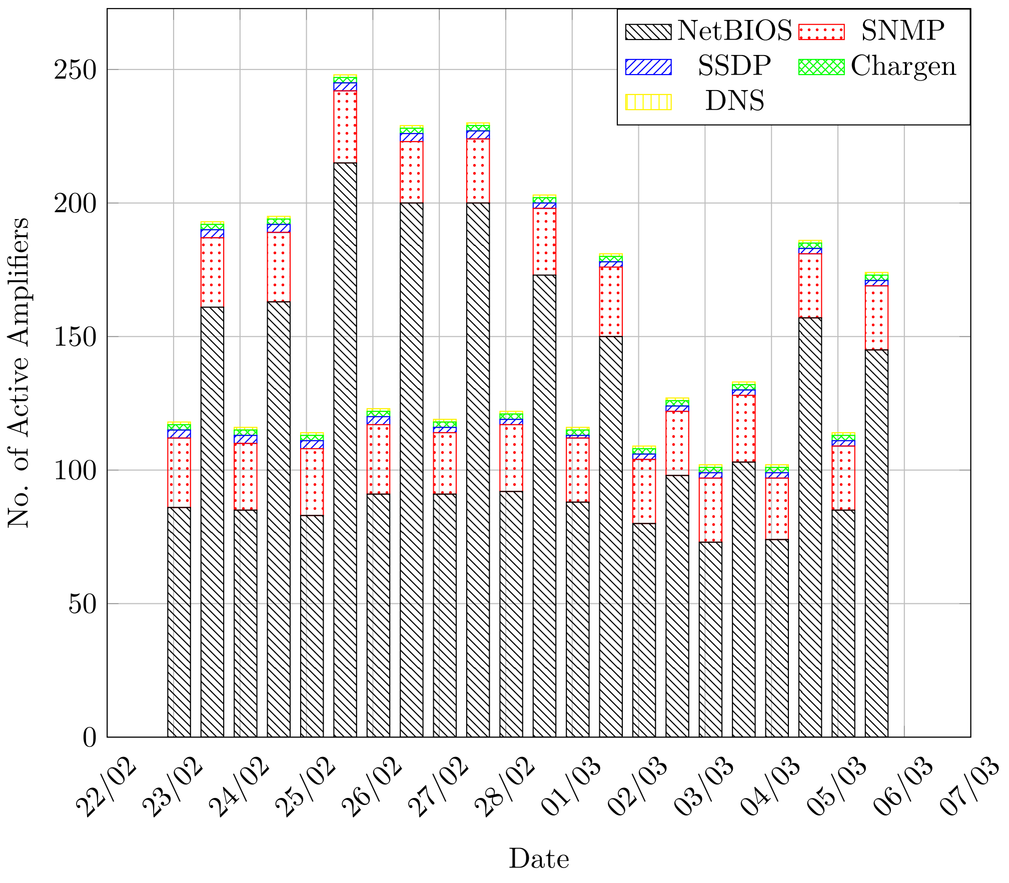


Figure 5.3: Number of active amplifiers over a period of two weeks.

It is evident from Figure 5.3 that the number of NetBIOS amplifiers follows a saw-tooth pattern. This is due to the fact that the results of scan execution during the working hours contains a large number of end user's personal computers and laptops running NetBIOS services. To verify this claim, we observed the addresses of NetBIOS amplifiers. The Figure 5.4 shows the number of NetBIOS amplifiers detected in three different subnets. Due to privacy reasons, we have not mentioned the network addresses, instead we refer them as *subnets*. The numbers of amplifiers in subnet-2 and subnet-3 remain constant as compared to the amplifiers in subnet-1.

By observing the open network ports on amplifiers in subnet-2 and subnet-3, we noticed that most of the amplifier hosts are printers, mail servers, and LDAP servers. We also searched the rDNS records, we found out that some of the machines in subnet-2 and subnet-3 were hosting webpages of research projects. rDNS records also revealed the management web interface of printers, the Figure A.4 shows the webpage for a Xerox printer. The amplifiers in subnet-1 are the end user workstations or laptops. Subnet-1 belongs to TUM Informatik and it could be seen that the scans performed during daytime have more amplifiers as compared to the scans performed at night. Another important observation is the drop in the number of NetBIOS amplifiers in subnet-1 during the weekends.

5.5.2 SNMP Amplifiers

The second highest number of amplifiers in our measurements belong to the SNMP protocol. SNMP *GetBulkRequest* was added in SNMP v2c, it is generally used by attackers for amplification attacks. The response from a *GetBulkRequest* contains several properties e.g. system description, up-time, device location, and contact details. Similar to SNMP v1, the SNMP v2c devices also respond to an SNMP *GetRequest* which is used to fetch a single property. Therefore, to increase the scan coverage for devices running older versions of SNMP, we decided to use the SNMP *GetRequest* instead of an SNMP *GetBulkRequest*.

In order to detect SNMP amplifiers, we used an SNMP *GetRequest* for system description property. There are two major reasons for selecting system description property. Firstly, it is set by default on most of the devices as compared to other properties such as the device's physical location, contact details etc. which needs to be set by administrators. Secondly, the system description string has a higher response size as compared to other properties such as the system uptime. A system description string is shown below, it contains the manufacturer's name, device model, and some other device related information.

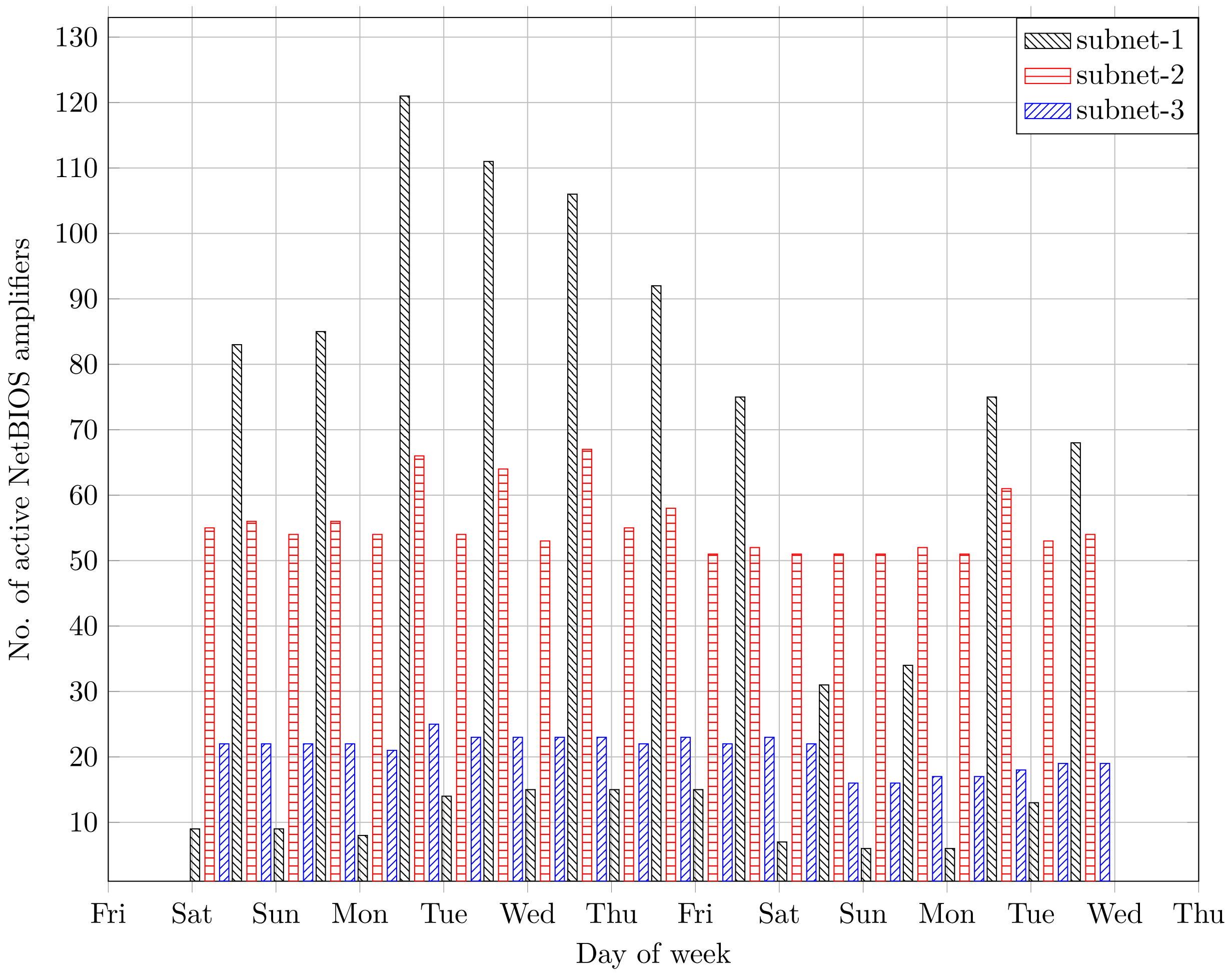


Figure 5.4: Number of active NetBIOS amplifiers in three subnets.

Vendor	**Device Type**	**Count**	**%**
HP	Printer	11	45.8
Brother	Printer	4	16.7
Kyocera	Printer	3	12.5
Samsung	Printer	2	8.3
Lexmark	Printer	1	4.2
Epson	External print server	1	4.2
nTop	nBox Netflow hardware appliance	1	4.2
APC	Uninterruptible Power Supply (UPS)	1	4.2

Table 5.3: SNMP amplifier device types and vendors.

```
Samsung Samsung M262x 282x Series; V3.00.01.04 JAN-18-2012;
Engine V1.00.02 01-17-2013;NIC V6.01.01;S/N ZD2JB8GD2B00LSR
```

Figure 5.5 shows the change in BAF of an SNMP amplifier during subsequent scans. We observed that the BAF of SNMP amplifiers remains constant because they always respond with the same system description string.

We detected a total of 24 SNMP amplifiers in our latest SNMP scan. Table 5.3 describes the vendors and the devices belonging to the SNMP amplifiers. All of these devices have a default passphrase *public* which makes these devices vulnerable for amplification attacks. 90% of the SNMP amplifiers that we detected are printers. It is interesting to note that SNMP amplifiers are mostly plug-and-play devices. Administrators usually ignore configuring the security settings on these devices such as changing the factory default passphrase.

5.5.3 SSDP Amplifiers

In our network measurements, we detected only 2 SSDP amplifiers. SSDP is used for the advertisement and discovery of UPnP devices in a network. The SSDP protocol is UDP based and it lacks the authentication mechanism. In order to discover UPnP devices, an SSDP search request is used which contains the *search target* (ST) parameter. The three options for the values of ST are listed below.

- UUID of a devices
- *upnp:rootdevice* — search for only the UPnP root devices
- *ssdp:all* — search for all root and embedded devices, and the services running on them.

We used the ST *ssdp:all* to achieve a high bandwidth amplification factor. As shown in the Table 5.4, the average BAF of SSDP amplifiers is 14.86. Upon taking a closer look at the amplifiers, we observed that both of the amplifiers are Samsung printers. The amplifiers belong to the same subnet, this indicates a missing rule in the firewall that has exposed the SSDP enabled devices to the public internet.

5.5.4 Chargen Amplifiers

We detected 2 Chargen amplifiers in our network measurements. The Chargen protocol is used to generate a character stream, it is considered a legacy protocol which is used for testing and debugging of networks. We achieved the highest BAF of 74 for the Chargen protocol. It should be noted that the Chargen protocol discards the contents of the request payload and responds with a fixed length string containing ASCII characters. Therefore, to achieve a high BAF, we used a single byte payload in our request packets.

We performed port scanning using Nmap on our detected Chargen amplifiers. We found out that both of the Chargen amplifiers were also running other legacy protocols such as Echo [14], Discard [8] and Daytime [7]. We observed that both amplifiers were running an outdated Sun Solaris 8 (SPARC) operating system which was released in February 2000 and its support was discontinued in March 2012. To support our claim of Sun Solaris OS running on the servers, we observed an open port 898 that is used by the Sun Management Console. The Figure A.5 shows the webpage for the Sun Management Console from one of the Chargen amplifiers. The Solaris 8 operating system enables the Chargen service by default [27]. We conclude that the legacy Chargen service was not installed by the user but shipped and enabled by an outdated OS.

5.5.5 DNS Amplifiers

We detected one DNS open resolver in our measurements. We used the DNS ANY query for google.com domain to generate a large response from the resolver. Table 5.4 shows that the average BAF of the DNS amplifier is 12.56. We observed that the BAF of DNS amplifier varies with time because the DNS resolver caches DNS query results for a configured period of time. Figure 5.5 shows the change in BAF of a DNS, SNMP and NetBIOS amplifier. It is evident that the BAF of the DNS amplifier drops during the weekend (1st - 3rd March) when the load on the resolver is low. We conclude that the open resolvers which are heavily utilized have significantly higher BAF as compared to the less utilized open resolvers.

Protocol	**Amplifiers**	**BAF**		
		all	**50%**	**10%**
NetBIOS	145	3.24	4.06	4.91
SNMP	24	3.02	4.15	6.47
SSDP	2	14.86	19.7	19.7
Chargen	2	74	74	74
DNS	1	12.56	12.56	12.56

Table 5.4: Table shows the Bandwidth Amplification Factor per protocol recorded from the last scanning iterations, *all* shows the average BAF of all amplifiers, 50% and 10% shows the average BAF of 50% and 10% of the worst amplifiers, respectively.

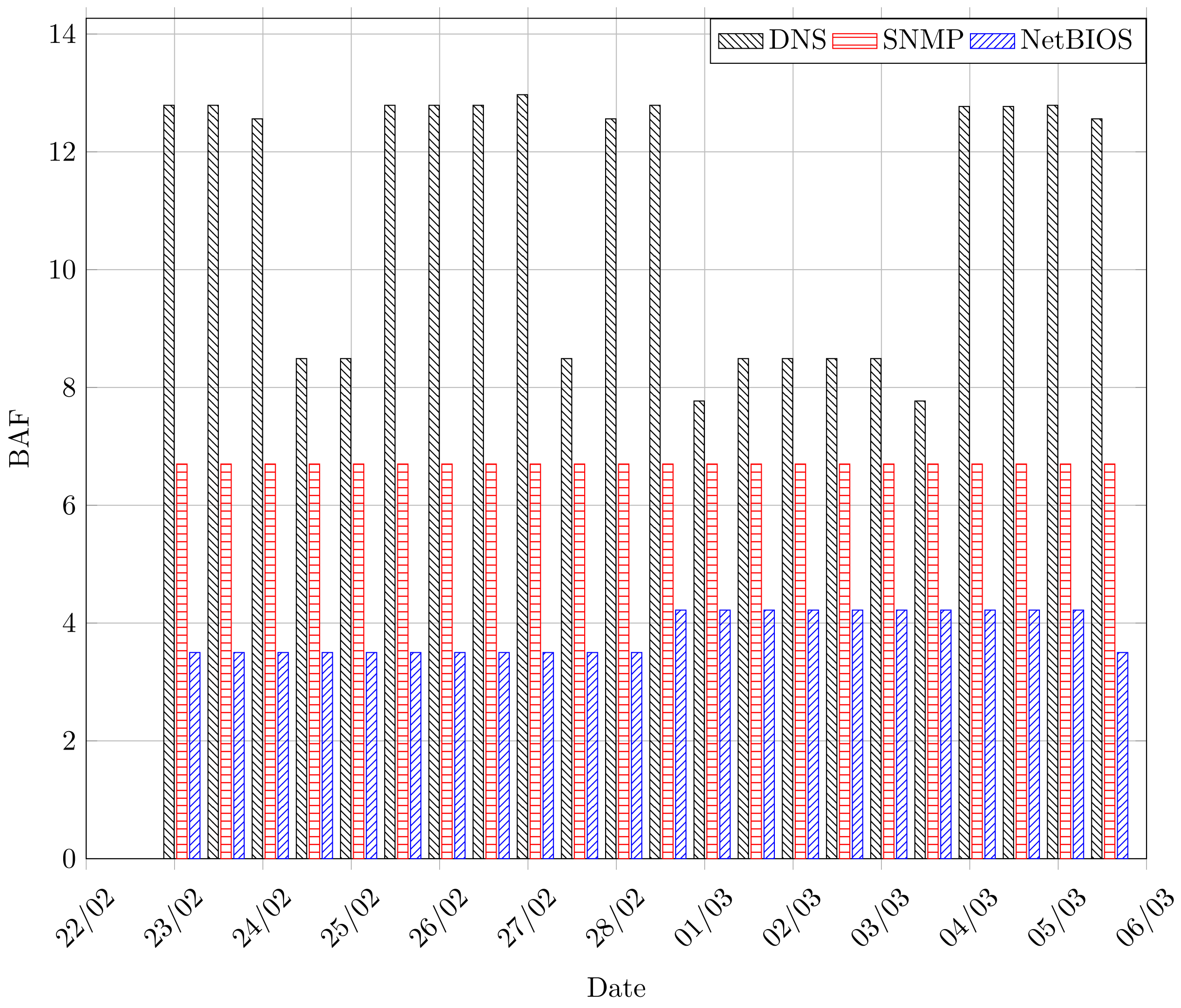


Figure 5.5: Change in the BAF of an amplifier belonging to DNS, SNMP and NetBIOS protocol respectively.

Pos.	Protocol Combination	Count
1	NetBIOS, SNMP	7
2	NetBIOS, SSDP	1
3	NetBIOS, SSDP, SNMP	1

Table 5.5: Combinations of vulnerable protocols detected in amplifiers.

5.5.6 Hosts with multiple amplifier services

In our network measurements, we detected that some of the hosts were running multiple services which could be abused for bandwidth amplification attacks. Table 5.5 shows the combinations of services running on the amplifiers. In order to find the device types, we used Nmap to analyze the services running on hosts by port scanning. Moreover, the SNMP amplifier's response containing the system description was also used to determine the device type. Our analysis showed that all of the devices with multiple amplification vulnerabilities are printers.

5.6 Load on the amplifier network

In this section, we describe the load on network hosts when they are scanned by the framework for amplifier detection. We have leveraged a horizontal scanning technique by using ZMap which scans of multiple hosts for the same port. Since we send one probe packet per host in a particular scan, the total number of probes packets sent to a host in an amplifier network in a day is the sum of the iterations performed by all scans every day. We scheduled 10 scans using our framework, each scan was executed twice a day. Hence, in our experimental setup, every host received 20 probe packets per day. Due to the use of horizontal scanning technique, the probe packets for a particular host are widely dispersed.

During the network measurements, there are chances of saturating the source and target networks with probe packets. This problem could be addressed by first using a low transmission rate, and secondly by avoiding sequentially probing the network addresses. In this context, we used a low transmission rate of 128 pps and the ZMap's optimized probing has helped in selecting addresses randomly to avoid saturating the network paths in the amplifier network.

5.7 Summary

The following list summarizes our observations.

1. We have not detected any publicly available amplifier for CLDAP, RIPv1, NTP, Memcached, and QOTD protocols in TUM's public IPv4 address space.
2. The maximum number of amplifiers in our measurements belong to NetBIOS.
3. The maximum BAF of 74 is recorded for the Chargen protocol.
4. Nine hosts in our network scans exhibited multiple amplification vulnerabilities.
5. The number of active NetBIOS amplifiers is affected by the diurnal usage pattern of the users.
6. The default security settings render devices to be abused for carrying out amplification attacks e.g. default passphrase *public* in case of SNMP.
7. A misconfigured firewall exposes devices running protocols that are designed to strictly operate in a local network for example SSDP, NetBIOS, SNMP etc.
8. The BAF for some protocols varies with time e.g. DNS open resolvers exhibited a lower BAF during the weekend when it faced less load from the users.
9. Legacy protocols which could be abused for amplification are sometimes enabled by default in old operating systems e.g. Chargen protocol is enabled by default in Solaris 8 OS [27].

Chapter 6

Related Work

In this chapter, we describe the related work in areas of different amplification attacks, amplifier detection and the mitigation strategies for DRDoS.

6.1 Amplification Attacks

Rossow [38] examined 14 UDP based protocols which are susceptible to bandwidth amplification attacks. For every protocol involved in his network measurements, he presented the total number of amplifiers, the amplification factors and the time it took to find 1000 and 100,000 amplifiers for each protocol. His network measurements have shown that the NTP protocol could be used to amplify the traffic up to a factor of 4670. He has demonstrated that there are a large number of amplifiers for legacy protocols such as Chargen and QOTD. His research also highlighted the amplification vulnerabilities in game protocols and P2P protocols. He has used bait services and NetFlow data from an ISP to identify the victims and amplifiers for bandwidth amplification attacks. Rossow has also presented countermeasures for DRDoS attacks and discussed the methods to harden vulnerable protocols. We have used a majority of protocols mentioned by Rossow in our network measurements.

Rossow has targetted SNMPv2 amplifiers using the *GetBulk* request, we have used an SNMP *GetRequest* to increase the scan coverage to SNMPv1 amplifiers in addition to the SNMPv2 amplifiers. Rossow has reported that the statistical *mode* for the response size from the Chargen amplifiers is 74 bytes. The Chargen amplifiers detected in our measurements conformed to Rossow's observation by responding with a 74 bytes response. Rossow has reported an average amplification factor of 28.7 and 30.8, in contrast, we have detected 12.56 and 14.86 for DNS open resolvers and SSDP amplifiers respectively. We attribute this significant skew from Rossow's measurements to the size of scanned networks. Our measurements were performed on a university's network whereas Rossow performed an internet-scale scan.

In our network measurements, we have followed an amplification threat model which involves only UDP based protocol. Kührer *et al.* [23] have shown that the connection-oriented TCP protocol could also be used to carry out amplification attacks. In case of TCP, no data is exchanged before the handshake is completed and all segments of the same size are exchanged during the handshake. So one might assume that TCP is not vulnerable to amplification attacks. Kührer *et al.* have demonstrated that a large number of TCP stacks re-transmit the SYN-ACK packet to the victim. They have reported that there are millions of systems with vulnerable TCP stacks that could amplify the TCP traffic by a factor of 20.

Naoumov and Ross [30] have shown that P2P protocols could also be abused for amplification. They have demonstrated that by poisoning the distributed index and routing table in the peers, the victim could be advertised as a legitimate peer hosting the most popular data files. In this way, the victim will receive a large number of TCP SYN packets from peers who want to download the files. The victim will also receive UDP based overlay maintenance messages from peers. Such unsolicited messages could exhaust the victim's bandwidth. Adamsky *et al.* have shown that the P2P file sharing system can also be abused for amplification by exploiting the two-way handshake of uTP which is the transport layer protocol for BitTorrent.

Gasser *et al.* [18] assessed the amplification vulnerabilities in building automation and control protocol (BACnet). Their internet-scale network scans discovered 16,485 BACnet devices, from which 14,000 devices were marked as vulnerable to amplification. They coordinated with a CERT to run a notification campaign to remediate the BACnet amplification threat. They reported the success in their notification campaign by showing the reduction in BACnet amplifiers. Our framework also supports a notification module to inform the network administrators about the amplification vulnerabilities in their networks.

6.2 Amplifier Detection

Rossow [38] has shown that the ratio between incoming and outgoing bytes from a network service can be used to classify a service as an amplifier. For successful classification, an amplifier has to exhibit a BAF of at least 5 and have sent at least 10MB of data to the victim. Instead of the passive approach, we have focused on the proactive approach for detecting amplifiers using active measurements.

The *Open Resolver Project* [34] and the *Open NTP Project* [33] aim to detect the vulnerable DNS resolvers and NTP servers respectively. Both of these projects perform active network measurements and provide a search utility for network administrators to check if the hosts in their networks were detected in the scans. These projects are focused on scanning the public IPv4 address space. In addition to the IPv4 address space, our framework is capable of executing the network scans on IPv6 addresses.

6.3 Mitigation Strategies

One of the key elements in the amplification threat model is source IP address spoofing. In this context, the Spoofer project [40] executes active network measurements to gather information about spoofable networks. The project relies on the participation of volunteers who install the program which tests the spoofability of the network and uploads the results to a server. The Spoofer project has recommended BCP38 to block IP packets with forged source addresses from entering the public internet.

Switching to a connection-oriented protocol to serve large responses is also considered as a solution to mitigate the amplification threat. Roland *et al.* [37] has shown that DNS responses of more than 512 bytes are generated mostly for *ANY* type query. It is recommended to use TCP for answers exceeding 512 bytes, thereby eliminating the amplification threat.

Rossow [38] has described several countermeasures for bandwidth amplification attacks. In addition to the control on source IP address spoofing and adding session handling to protocols, he describes increasing the size of request packets to maintain request/response size symmetry. He considers it as an inadequate solution because it would mitigate the amplification threat at the cost of a drop in the efficiency of a protocol.

Rossow evaluated the solution of rate limiting the number of requests and concludes that this solution would only help if a single amplifier is used in bandwidth amplification attacks. He has argued that an attacker could lower the request rate and use a large number of amplifiers to achieve a high attack bandwidth. He has also described the importance of secure service configuration. We have also concluded from the analysis of our scanning results, that a misconfigured firewall could expose the network services such as the DNS resolvers to the public internet. Furthermore, the default passphrase on

SNMP-enabled devices could render them as attractive amplifiers for attackers. Hence, the network administrators should pay attention to securely configuring services.

Chapter 7

Conclusion

In this chapter, we discuss the measures that can be taken to protect the services against amplification abuse. We also describe the countermeasures which can reduce the number of globally available amplifiers. Moreover, in this chapter, we present an outlook on the future work and conclude this thesis with a summary.

7.1 Countermeasures

Following proactive countermeasures can be taken to mitigate the abuse of services for amplification.

1. Prevent IP address spoofing
   IP address spoofing plays a major role in carrying out a DRDoS attack. If IP address spoofing is made impossible then none of the services could be abused for amplification because the attacker won't be able to backscatter the traffic to the victim. As of this writing, the Spoofer project states that 20% of the autonomous systems are vulnerable to IP address spoofing. We urge the network administrators to configure their firewalls according to BCP38 to play their part in the mitigation of IP address spoofing.

2. Prevent public access to devices
   One import characteristic of amplifiers is that they are publicly available on the Internet. Protocols such as SNMP, SSDP, NetBIOS etc. are strictly designed for usage in local area networks. Similarly, the DNS resolvers should perform name resolution only for private hosts. Therefore, the administrators should configure firewalls to block the access of such hosts from the public internet.

3. Properly configure services
   A large number of devices could be saved from amplification abuse if they are configured properly. For example, the factory default passphrase should be changed on SNMP enabled devices, the *monlist* command should be disabled on the NTP servers and the protocols such as Chargen should be disabled on the older operating systems such as Solaris 8 OS. In our network measurements, we have observed that a large number of amplifiers are plug-and-play devices. Therefore, we recommend to pay special attention in configuring them.

7.2 Future work

Our network measurements were restricted to IP addresses of TUM. In the future, we will provide the framework to LRZ to help them to detect amplifiers in MWN. When a new amplification vulnerability will be discovered, we will add the corresponding attack vectors in our framework to scan for the latest amplification vulnerabilities. We will conduct internet-wide measurements to find the most vulnerable protocols on a global scale. Our internet-wide network scans would also reveal the geographical distribution of amplifiers. In addition to scanning the IPv4 address space, we will conduct large scale scans on IPv6 address space using a comprehensive hitlit [17]. We will also share our results from Internet-wide scans in a secure way so that the network administrators can be notified about the services in their networks which can be abused for amplification.

7.3 Summary

We developed a framework for the detection of amplifiers using active measurements. The framework can not only be used to detect amplification vulnerabilities in private networks, but it is also capable of performing internet-wide scans for research purposes. We demonstrated the effectiveness of our framework by performing scans in a university's network. Our scanning results revealed that the number of amplifiers is affected by the diurnal usage pattern of the users and the amplification factor for some protocols varies with time. Our scans have uncovered misconfigurations in firewalls and network hosts. We also discussed solutions for protecting network hosts from being abused for amplification.

Chapter A

Appendix

A.1 CLI Client Usage

The following commands describe the usage of CLI client.

Client Usage:

```
$ python scan.py --help
usage: scan.py [-h] [--name NAME] [--rate RATE] [--target-port TARGET_PORT]
               [--target-hosts TARGET_HOSTS [TARGET_HOSTS ...]]
               [--target-hosts-file TARGET_HOSTS_FILE]
               [--request-payload REQUEST_PAYLOAD] [--minute MINUTE]
               [--hour HOUR] [--day-of-week DAY_OF_WEEK]
               [--day-of-month DAY_OF_MONTH] [--month-of-year MONTH_OF_YEAR]
               [--latest LATEST] [--task-id TASK_ID] [--raw]
               {create,delete,update,list,info,result,disable,enable,list-running,kill}

positional arguments:
  {create,delete,update,list,info,result,disable,enable,list-running,kill}

optional arguments:
  -h, --help            show this help message and exit
  --name NAME           set name of scan
  --rate RATE           Set send rate in packets/sec
  --target-port TARGET_PORT
                        port number to scan
  --target-hosts TARGET_HOSTS [TARGET_HOSTS ...]
                        Space separated list of target address ranges in CIDR
                        notaion
  --target-hosts-file TARGET_HOSTS_FILE
                        path to file containing address ranges, one on every
                        line
  --request-payload REQUEST_PAYLOAD
                        HEX encoded request payload
  --minute MINUTE       Minute field for Cron
  --hour HOUR           Hour field for Cron
  --day-of-week DAY_OF_WEEK
                        Day of the week field for Cron
  --day-of-month DAY_OF_MONTH
```

```
                        Day of the month field for Cron
  --month-of-year MONTH_OF_YEAR
                        Month of the year field for Cron
  --latest LATEST       Get n number of latest scan results
  --task-id TASK_ID     Task ID of the scan job
  --raw                 print raw json data

\end{verbatim}

\textbf{Create Command:}
\begin{verbatim}
$ python scan.py  create --name scan_two \
                                --target-port 12 \
                                --target-hosts 10.10.0.0/16 192.168.0.0/16 \
                                --request-payload abcabcabc \
                                --minute "*/2" \
                                --rate 128

Scan 'scan_two' created successfully
```

Info Command:

```
$ python scan.py info --name scan_two

Details for scan 'scan_two'


            * Enabled: True
            * Total run count: 4
            * Last run at: 2019-01-22T11:04:00.005508Z
            * Target addresses: ['10.10.0.0/16', '192.168.0.0/16']
            * Target port: 12
            * Request Hexdump: abcabcabc
            * Cron: */2 * * * *
            * Rate: 128
```

List Command:

```
$ python scan.py list

Scans List:

* scan_two
* scan_one
```

Result Command:

```
$ python scan.py result --name scan_anonymous


* Status: SUCCESS
* Created: 2019-01-22T11:04:00.042167Z
* Number of Amplifiers: 2
+-----------------+---------------+---------------------+
|    Amplifier    | Response Size | Amplification Factor |
+-----------------+---------------+---------------------+
| XXX.XXX.XXX.XXX |      1772     |         37.7         |
|  XXX.XXX.XXX.XX |      553      |        11.77         |
+-----------------+---------------+---------------------+
```

Get N Latest Scanning Results:

```
$ python scan.py result --name scan_anonymous --latest=2


* Status: SUCCESS
* Created: 2019-01-22T11:02:00.037994Z
* Number of Amplifiers: 3
+-----------------+---------------+----------------------+
|    Amplifier    | Response Size | Amplification Factor |
+-----------------+---------------+----------------------+
|  XXX.XX.XX.XXX  |      9968     |        284.8         |
|  XXX.XX.XXX.XXX |      9398     |        268.51        |
|  XX.XX.XXX.XXX  |      7467     |        213.34        |
+-----------------+---------------+----------------------+
* Status: SUCCESS
* Created: 2019-01-22T11:04:00.042167Z
* Number of Amplifiers: 2
+-----------------+---------------+----------------------+
|    Amplifier    | Response Size | Amplification Factor |
+-----------------+---------------+----------------------+
| XXX.XXX.XXX.XXX |      1772     |         37.7         |
|  XXX.XXX.XXX.XX |      553      |        11.77         |
+-----------------+---------------+----------------------+
```

Enable & Disable Command:

```
$ python scan.py disable --name scan_one
Scan 'scan_one' disabled successfully

$ python scan.py enable --name scan_one
Scan 'scan_one' enabled successfully
```

Update Command:

```
$ python scan.py update --name scan_one --target-hosts 10.0.0.0/16 \
                                        --target-port 54 \
                                        --request-payload aaaaa \
                                        --minute "*/3" \
                                        --rate 256
Scan 'scan_one' updated successfully
```

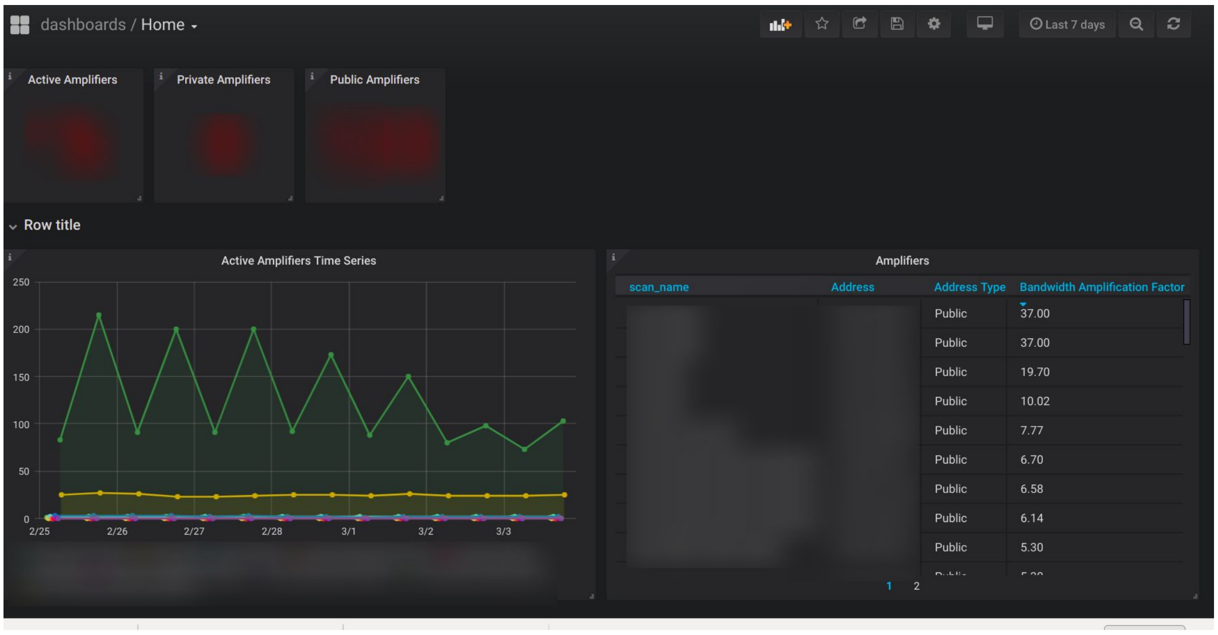
Killing a running scan:

```
$ python scan.py list-running

Task ID: e6bde8ae-1ba2-4313-b766-a0bcb1dbc225
Scan Args: {'scan_name': 'scan_two',
            'address_range': ['10.10.0.0/24', '10.0.0.0/8'],
            'target_port': 23,
            'request_hexdump': 'abcabcabc',
            'cron_str': '* * * * *',
            'version': 4,
            'rate': 256}

$ python scan.py kill --task-id e6bde8ae-1ba2-4313-b766-a0bcb1dbc225
Task 'e6bde8ae-1ba2-4313-b766-a0bcb1dbc225' is killed successfully
```

Delete Command:

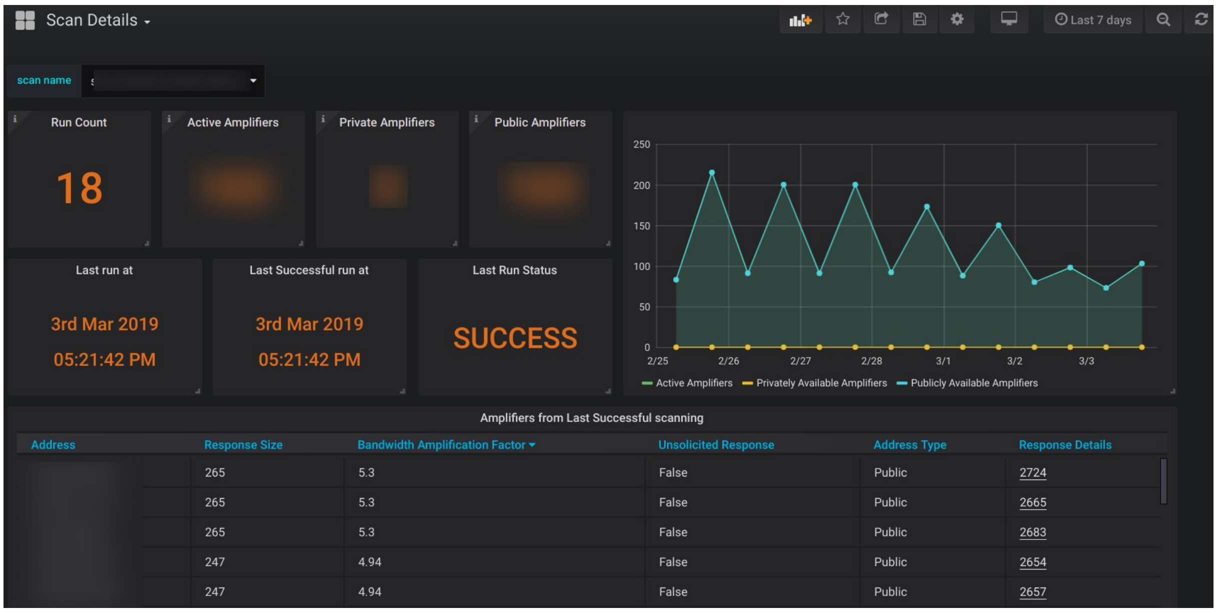
```
$ python scan.py delete --name scan_two
Scan 'scan_two' deleted successfully
```

A.2 Grafana Dashboards



scan_name	Address	Address Type	Bandwidth Amplification Factor
[redacted]	[redacted]	Public	37.00
[redacted]	[redacted]	Public	37.00
[redacted]	[redacted]	Public	19.70
[redacted]	[redacted]	Public	10.02
[redacted]	[redacted]	Public	7.77
[redacted]	[redacted]	Public	6.70
[redacted]	[redacted]	Public	6.58
[redacted]	[redacted]	Public	6.14
[redacted]	[redacted]	Public	5.30

Figure A.1: Home dashboard showing general information about scans.



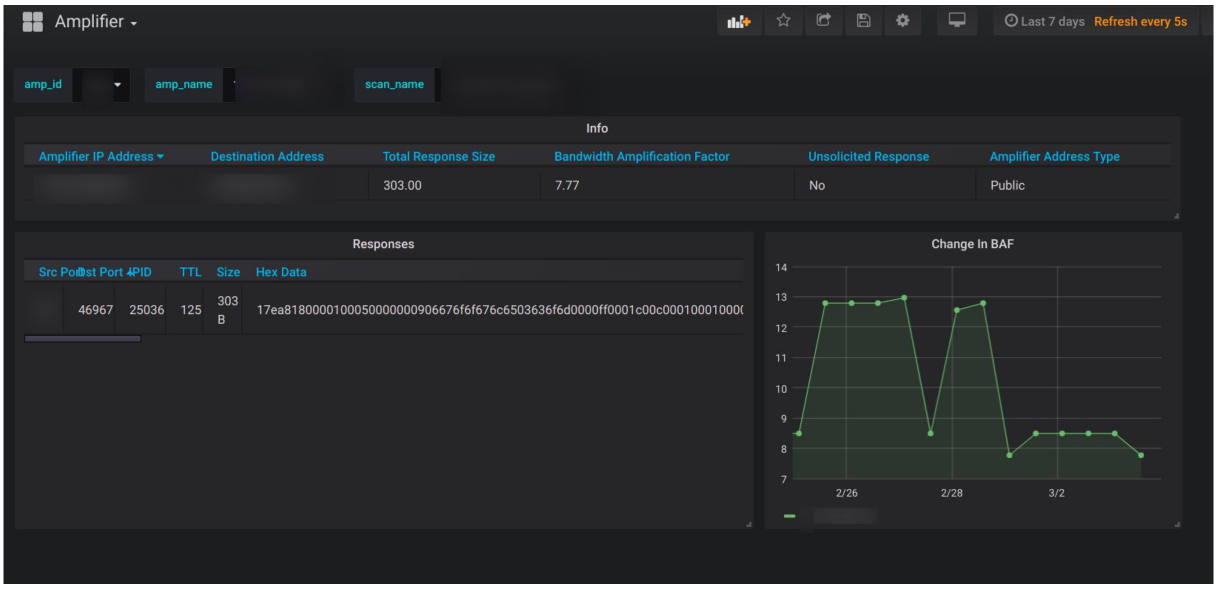
Run Count	Active Amplifiers	Private Amplifiers	Public Amplifiers
18	[redacted]	[redacted]	[redacted]

Last run at	Last Successful run at	Last Run Status
3rd Mar 2019 05:21:42 PM	3rd Mar 2019 05:21:42 PM	SUCCESS

Amplifiers from Last Successful scanning

Address	Response Size	Bandwidth Amplification Factor	Unsolicited Response	Address Type	Response Details
[redacted]	265	5.3	False	Public	2724
[redacted]	265	5.3	False	Public	2665
[redacted]	265	5.3	False	Public	2683
[redacted]	247	4.94	False	Public	2654
[redacted]	247	4.94	False	Public	2657

Figure A.2: Scan details dashboard showing specific details about a scan.



Info

Amplifier IP Address	Destination Address	Total Response Size	Bandwidth Amplification Factor	Unsolicited Response	Amplifier Address Type
[redacted]	[redacted]	303.00	7.77	No	Public

Responses

Src Port	Dst Port	PID	TTL	Size	Hex Data
[redacted]	46967	25036	125	303 B	17ea8180000100050000000906676f6f676c6503636f6d0000ff0001c00c00010001000(

Figure A.3: Amplifier dashboard showing information about a specific amplifier.

A.3 Response Packets

A.3.1 NetBIOS

A response packet received form an NBSTAT list names query is shown below.

```
Internet Protocol Version 4, Src: X.X.X.X, Dst: X.X.X.X
User Datagram Protocol, Src Port: 137, Dst Port: 43411
NetBIOS Name Service
    Transaction ID: 0xe5d8
    Flags: 0x8400, Response, Opcode: Name query, Authoritative, Reply code: No error
    Questions: 0
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
    Answers
        *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>: type NBSTAT, class IN
            Name: *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
            Type: NBSTAT (33)
            Class: IN (1)
            Time to live: 0 seconds
            Data length: 209
            Number of names: 9
            Name: SERVOJ<00> (Workstation/Redirector)
            Name flags: 0x6400, ONT: Unknown, Name is active
            Name: SERVOJ<03> (Messenger service/Main name)
            Name flags: 0x6400, ONT: Unknown, Name is active
            Name: SERVOJ<20> (Server service)
            Name flags: 0x6400, ONT: Unknown, Name is active
            Name: <01><02>__MSBROWSE__<02><01> (Browser)
            Name flags: 0xe400, Name type, ONT: Unknown, Name is active
            Name: FORSTSTUD<1d> (Local Master Browser)
            Name flags: 0x6400, ONT: Unknown, Name is active
            Name: FORSTSTUD<1b> (Domain Master Browser)
            Name flags: 0x6400, ONT: Unknown, Name is active
            Name: FORSTSTUD<1c> (Domain Controllers)
            Name flags: 0xe400, Name type, ONT: Unknown, Name is active
            Name: FORSTSTUD<1e> (Browser Election Service)
            Name flags: 0xe400, Name type, ONT: Unknown, Name is active
            Name: FORSTSTUD<00> (Workstation/Redirector)
            Name flags: 0xe400, Name type, ONT: Unknown, Name is active
            Unit ID: 00:00:00_00:00:00 (00:00:00:00:00:00)
            Jumpers: 0x00
            Test result: 0x00
            Version number: 0x0000
            Period of statistics: 0x0000
            Number of CRCs: 0
            Number of alignment errors: 0
            Number of collisions: 0
            Number of send aborts: 0
            Number of good sends: 0
            Number of good receives: 0
            Number of retransmits: 0
            Number of no resource conditions: 0
            Number of command blocks: 0
            Number of pending sessions: 0
            Max number of pending sessions: 0
            Max total sessions possible: 0
            Session data packet size: 0
```

A.3.2 QOTD

```
$ nc -u djxmmx.net  17
"Man␣can␣climb␣to␣the␣highest␣summits,␣but␣he␣cannot␣dwell␣there␣long."
 George Bernard Shaw (1856-1950)
```

A.3.3 CharGen

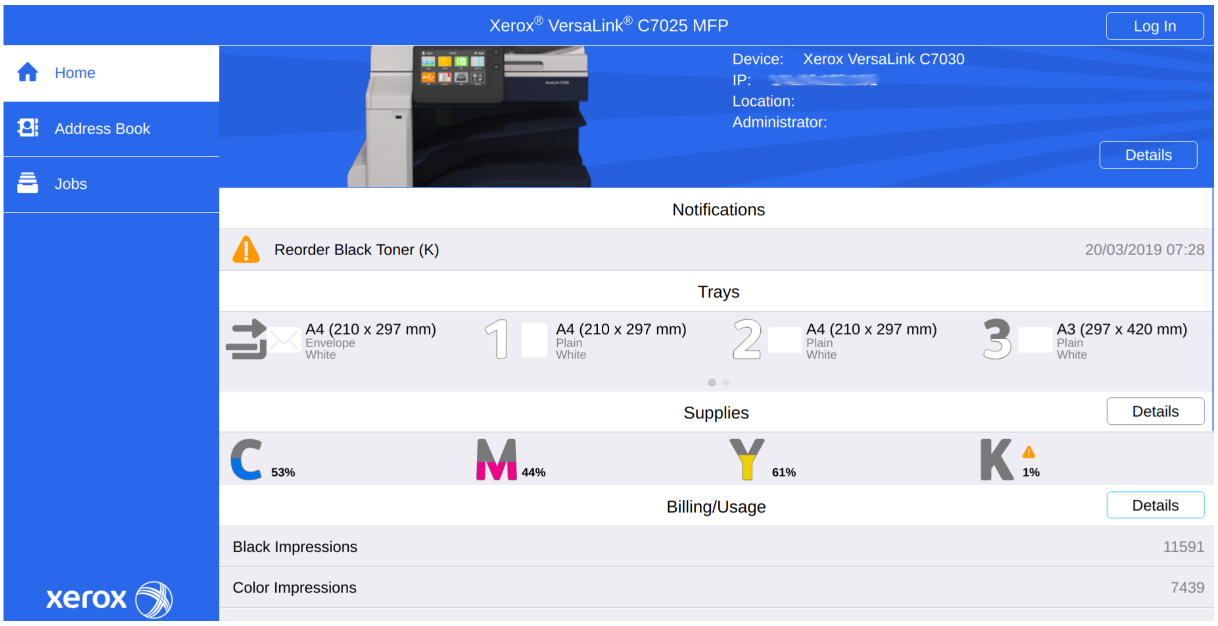
```
$ nc -u <ip-address>  19
789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
```

A.3.4 SNMP

The packet below shows the response for a SNMP get system description request.

```
Frame 15: 332 bytes on wire (2656 bits), 332 bytes captured (2656 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: X.X.X.X, Dst: X.X.X.X
User Datagram Protocol, Src Port: 161, Dst Port: 34873
Simple Network Management Protocol
    version: version-1 (0)
    community: public
    data: get-response (2)
        get-response
            request-id: 1448795229
            error-status: noError (0)
            error-index: 0
            variable-bindings: 1 item
                1.3.6.1.2.1.1.1.0: 5865726f7820576f726b43656e74726520373833302
                    Object Name: 1.3.6.1.2.1.1.1.0 (iso.3.6.1.2.1.1.1.0)
                    Value (OctetString): 5865726f7820576f726b43656e7472652037383330
                        Variable-binding-string [truncated]: Xerox WorkCentre 7830 v1;
                        SS 072.010.165.14201, NC 072.015.14201, UI 072.015.14201,
                        ME 090.029.000, CC 072.015.14201, DF 012.003.000,
                        FI 007.002.000, FA 003.011.021, CCOS 072.005.14201,
                        NCOS 072.005.14
```

A.4 MISCELLANEOUS


Xerox® VersaLink® C7025 MFP

Log In

Home

Address Book

Jobs

Device: Xerox VersaLink C7030
IP: [redacted]
Location:
Administrator:

Details

Notifications

Reorder Black Toner (K) 20/03/2019 07:28

Trays

A4 (210 x 297 mm)
Envelope
White

1 A4 (210 x 297 mm)
Plain
White

2 A4 (210 x 297 mm)
Plain
White

3 A3 (297 x 420 mm)
Plain
White

Supplies Details

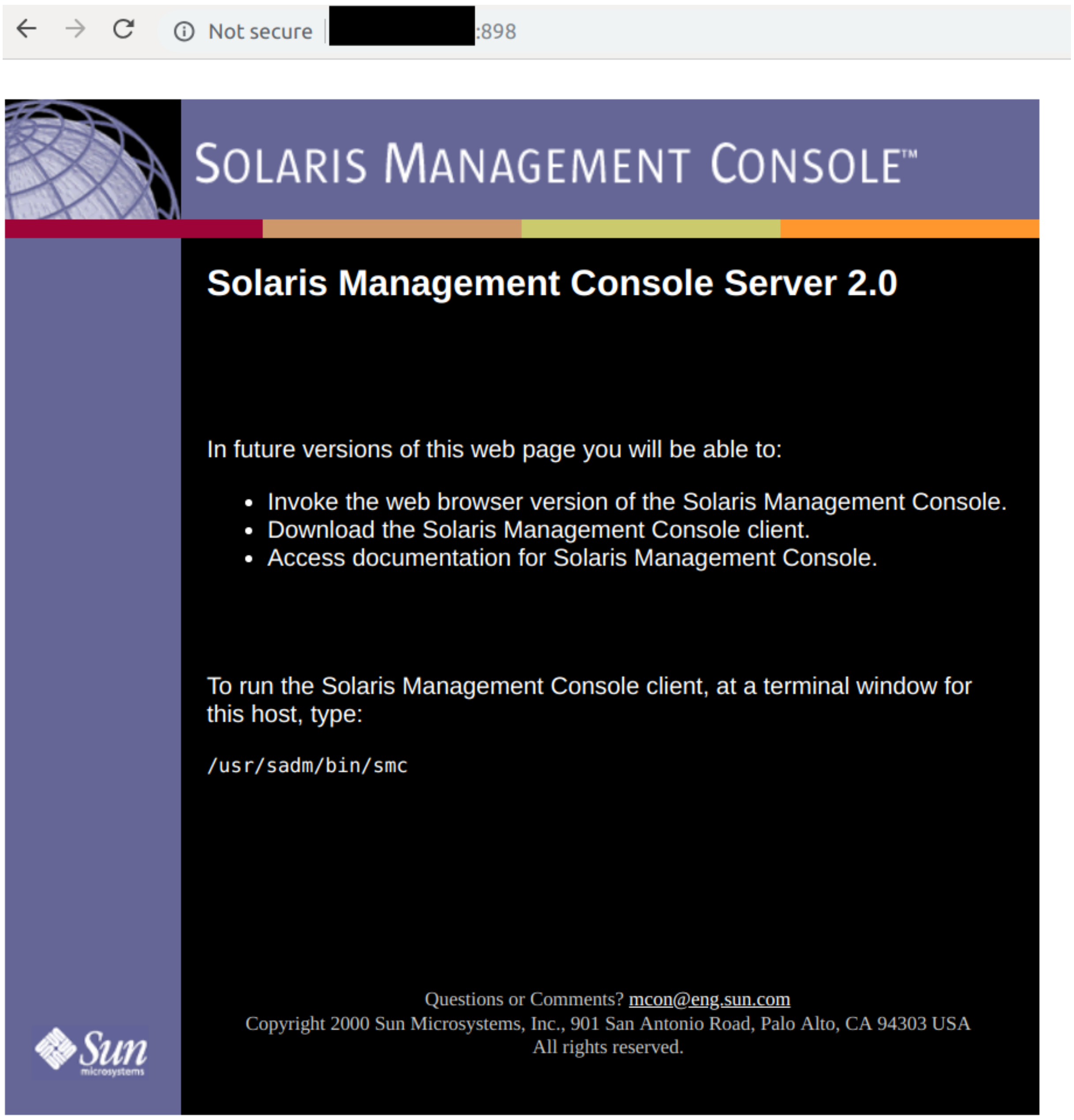
C 53% M 44% Y 61% K 1%

Billing/Usage Details

Black Impressions	11591
Color Impressions	7439

xerox

FIGURE A.4: Web interface of a Xerox printer.


Not secure [redacted]:898

SOLARIS MANAGEMENT CONSOLE™

Solaris Management Console Server 2.0

In future versions of this web page you will be able to:

- Invoke the web browser version of the Solaris Management Console.
- Download the Solaris Management Console client.
- Access documentation for Solaris Management Console.

To run the Solaris Management Console client, at a terminal window for this host, type:

```
/usr/sadm/bin/smc
```

Questions or Comments? mcon@eng.sun.com
Copyright 2000 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA
All rights reserved.

Figure A.5: Sun Management Console.

Chapter B

List of acronyms

BAF	Bandwidth amplification factor.
CLDAP	Connection-less lightweight directory access protocol.
CLI	Command line interface.
DNS	Domain name system.
DoS	Denial of service.
DRDoS	Distributed reflection denial of service.
DRF	Django rest framework.
IEEE	Institute of electrical and electronics engineers.
IoT	Internet of things.
ISP	Internet service provider.
LDAP	Lightweight directory access protocol.
LRZ	Leibniz supercomputing centre.
MVC	Model view controller.
MWN	Munich scientific network.
NetBIOS	Network basic input/output system.
NTP	Network time protocol.
OSI	Open Systems Interconnection. Reference model for layered network architectures by the OSI.
PPS	Packets per second.
QOTD	Quote of the day.

REST	Representational state transfer.
RIP	Routing information protocol.
RR	Resource record.
SNMP	Simple network management protocol.
SSDP	Simple service discovery protocol.
TCP	Transmission control protocol. Stream-oriented, reliable, transport layer protocol.
UDP	User datagram protocol. Datagram-oriented, unreliable transport layer protocol.
UPnP	Universal plug-and-play.

Bibliography

[1] Roy Arends et al. "RFC 4033: DNS security introduction and requirements". In: *Updated by RFC6014, RFC6840* (2005).

[2] Scott O. Bradner and Joel M. Halpern. *RIPv1 Applicability Statement for Historic Status*. RFC 1923. Mar. 1996. DOI: `10.17487/RFC1923`. URL: `https://rfc-editor.org/rfc/rfc1923.txt`.

[3] Timm Böttger et al. "DoS Amplification Attacks – Protocol-Agnostic Detection of Service Abuse in Amplifier Networks". In: Apr. 2015, pp. 205–218. ISBN: 978-3-319-17171-5. DOI: `10.1007/978-3-319-17172-2_14`.

[4] Josiah L. Carlson. *Redis in Action*. Greenwich, CT, USA: Manning Publications Co., 2013. ISBN: 1617290858, 9781617290855.

[5] Stephen Cass. *The 2018 Top Programming Languages*. 2018. URL: `https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages`.

[6] *Character Generator Protocol*. RFC 864. May 1983. DOI: `10.17487/RFC0864`. URL: `https://rfc-editor.org/rfc/rfc864.txt`.

[7] *Daytime Protocol*. RFC 867. May 1983. DOI: `10.17487/RFC0867`. URL: `https://rfc-editor.org/rfc/rfc867.txt`.

[8] *Discard Protocol*. RFC 863. May 1983. DOI: `10.17487/RFC0863`. URL: `https://rfc-editor.org/rfc/rfc863.txt`.

[9] *Distributed Task Queue*. URL: `http://www.celeryproject.org/`.

[10] *Django*. URL: `https://www.djangoproject.com/`.

[11] *Django REST Framework*. URL: `https://www.django-rest-framework.org/`.

[12] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. "ZMap: Fast Internet-wide Scanning and Its Security Applications". In: *Proceedings of the 22Nd USENIX Conference on Security*. SEC'13. Washington, D.C.: USENIX Association, 2013, pp. 605–620. ISBN: 978-1-931971-03-4.

[13] *Dyn Analysis Summary Of Friday October 21 Attack | Dyn Blog*. URL: `https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/`.

[14] *Echo Protocol.* RFC 862. May 1983. DOI: 10.17487/RFC0862. URL: https://rfc-editor.org/rfc/rfc862.txt.

[15] P Ferguson and D Senie. *RFC2827 (BCP38): Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. IETF, May 2000.*

[16] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". AAI9980887. PhD thesis. 2000. ISBN: 0-599-87118-0.

[17] Oliver Gasser et al. "Scanning the IPv6 Internet: Towards a Comprehensive Hitlist". In: *Proc. 8th Int. Workshop on Traffic Monitoring and Analysis.* Louvain-la-Neuve, Belgium, Apr. 2016. URL: https://net.in.tum.de/pub/ipv6-hitlist/.

[18] Oliver Gasser et al. "The Amplification Threat Posed by Publicly Reachable BACnet Devices". In: *Journal of Cyber Security and Mobility* (Oct. 2017). URL: https://www.riverpublishers.com/journal_read_html_article.php?j=JCSM/6/1/4.

[19] *GitHub.* URL: https://github.com/.

[20] *Grafana - The open platform for analytics and monitoring.* URL: https://grafana.com/.

[21] *Internet Protocol.* RFC 791. Sept. 1981. DOI: 10.17487/RFC0791. URL: https://rfc-editor.org/rfc/rfc791.txt.

[22] *IPv6 probe for ZMap.* 2019. URL: https://github.com/tumi8/zmap.

[23] Marc Kührer et al. "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks". In: *23rd USENIX Security Symposium (USENIX Security 14).* San Diego, CA: USENIX Association, 2014, pp. 111–125. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kuhrer.

[24] Avraham Leff and James T Rayfield. "Web-application development using the model/view/controller design pattern". In: *Proceedings fifth ieee international enterprise distributed object computing conference.* IEEE. 2001, pp. 118–127.

[25] *Memcached - a distributed memory object caching system.* URL: https://memcached.org/.

[26] David L Mills. "Internet time synchronization: the network time protocol". In: *IEEE Transactions on communications* 39.10 (1991), pp. 1482–1493.

[27] Ed Mitchell et al. *Hack Proofing Sun Solaris 8.* Syngress Publishing, 2001.

[28] Paul Mockapetris. "RFC-1034 Domain Names-Concepts and Facilities". In: *Network Working Group* (1987), p. 55.

[29] *MWN Network Topology.* URL: https://www.lrz.de/services/netz/mwn-ueberblick/Verkabelung.png.

[30] Naoum Naoumov and Keith Ross. “Exploiting P2P systems for DDoS attacks”. In: *Proceedings of the 1st international conference on Scalable information systems.* ACM. 2006, p. 47.

[31] *NETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack; The Terabit Attack Era Is Upon Us.* URL: `https://www.netscout.com/blog/asert/netscout-arbor-confirms-17-tbps-ddos-attack-terabit-attack-era`.

[32] *Nmap.* URL: `https://nmap.org/`.

[33] *NTP Scanning Project.* URL: `http://openntpproject.org/`.

[34] *Open Resolver Project.* URL: `http://openresolverproject.org/`.

[35] *Overview of the Munich Scientific Network (MWN).* URL: `https://www.lrz.de/services/netz/mwn-ueberblick_en/`.

[36] *Quote of the Day Protocol.* RFC 865. May 1983. DOI: `10.17487/RFC0865`. URL: `https://rfc-editor.org/rfc/rfc865.txt`.

[37] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. “DNSSEC and its potential for DDoS attacks: a comprehensive measurement study”. In: *Proceedings of the 2014 Conference on Internet Measurement Conference.* ACM. 2014, pp. 449–460.

[38] Christian Rossow. “Amplification Hell: Revisiting Network Protocols for DDoS Abuse”. In: *In Proceedings of the 2014 Network and Distributed System Security Symposium, NDSS.* 2014.

[39] Skottler. *February 28th DDoS Incident Report.* 2018. URL: `https://githubengineering.com/ddos-incident-report/`.

[40] *Spoofer project.* URL: `https://www.caida.org/projects/spoofer/`.

[41] *Standard Windows interoperability suite of programs for Linux and Unix (SAMBA).* URL: `https://www.samba.org/`.

[42] *The Spamhaus Project.* URL: `https://www.spamhaus.org/`.

[43] *TUM internet-wide scans abuse contact.* URL: `https://www.net.in.tum.de/projects/gino/index.html#internet-wide-scans`.

[44] Ihsan Ullah, Naveed Khan, and Hatim A Aboalsamh. “Survey on botnet: Its architecture, detection, prevention and mitigation”. In: *2013 10th IEEE International Conference on Networking, Sensing and Control (ICNSC).* IEEE. 2013, pp. 660–665.

[45] Alvaro Videla and Jason JW Williams. *RabbitMQ in action: distributed messaging for everyone.* Manning, 2012.

[46] Paul A. Vixie. *Extension Mechanisms for DNS (EDNS0).* RFC 2671. Aug. 1999. DOI: `10.17487/RFC2671`. URL: `https://rfc-editor.org/rfc/rfc2671.txt`.